

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: January 17, 2013

E. Abdo  
M. Boucadair  
J. Queiroz  
France Telecom  
July 16, 2012

HOST\_ID TCP Options: Implementation & Preliminary Test Results  
draft-abdo-hostid-tcpopt-implementation-03

Abstract

This memo documents the implementation of the HOST\_ID TCP Options. It also discusses the preliminary results of the tests that have been conducted to assess the technical feasibility of the approach as well as its scalability. Several HOST\_ID TCP options have been implemented and tested.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 17, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as

described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	4
2. Objectives . . . . .	4
3. NAT Reveal TCP Options: Overview . . . . .	5
3.1. HOST_ID_WING TCP Option . . . . .	5
3.2. HOST_ID_BOUCADAIR TCP Option . . . . .	5
3.2.1. SYN Mode . . . . .	6
3.2.2. ACK Mode . . . . .	7
4. Overview of the Linux Kernel Modifications . . . . .	7
5. Testbed Setup & Configuration . . . . .	8
5.1. Automated TCP Traffic Generator . . . . .	10
5.2. Testing Methodology and Procedure . . . . .	10
5.3. Check HOST_ID TCP Options are Correctly Injected . . . . .	11
5.4. Top Site List . . . . .	11
6. Experimentation Results . . . . .	11
6.1. HTTP Experimentation Results . . . . .	11
6.1.1. Configuration 1: Connected to an enterprise network . . . . .	12
6.1.1.1. Results . . . . .	12
6.1.1.2. Analysis . . . . .	14
6.1.2. Configuration 2: In a lab behind a firewall . . . . .	15
6.1.3. Configuration 3: Connected to two commercial ISP networks . . . . .	15
6.1.4. Additional Results . . . . .	16
6.1.5. Analysis . . . . .	16
6.2. FTP . . . . .	17
6.3. SSH . . . . .	18
6.4. Telnet . . . . .	18
7. AFTR Module Modifications . . . . .	19
7.1. Specification . . . . .	19
7.2. Verification . . . . .	20
7.3. CGN Performance Testing . . . . .	21
7.3.1. Configuration . . . . .	21
7.3.2. HTTP Testing . . . . .	22
7.3.2.1. Analysis of results . . . . .	24
7.3.2.2. Conclusion . . . . .	24
7.3.3. FTP . . . . .	25
8. IPTABLES: Modifications to Enforce Policies at the Server Side . . . . .	25
8.1. Overview . . . . .	25
8.2. Validation . . . . .	26
8.3. Stripping HOST_ID Options . . . . .	26
8.4. Logging a Specific HOST_ID Option Value . . . . .	27
8.5. Dropping a specific HOST_ID Option Value . . . . .	28
9. IANA Considerations . . . . .	29

10. Security Considerations . . . . .	29
11. Acknowledgments . . . . .	29
12. References . . . . .	29
12.1. Normative References . . . . .	29
12.2. Informative References . . . . .	29
Authors' Addresses . . . . .	30

## 1. Introduction

To ensure IPv4 service continuity, service providers will need to deploy IPv4 address sharing techniques. Several issues are likely to be encountered (refer to [RFC6269] for a detailed survey of the issues) and they may affect the delivery of services that depends on the enforcement of policies based upon the source IPv4 address.

Some of these issues may be mitigated owing to the activation of advanced features. Among the solutions analyzed in [I-D.boucadair-intarea-nat-reveal-analysis], the use of a new TCP option to convey a HOST\_ID seems to be a promising solution.

This memo documents some implementation and experimentation efforts that have been conducted to assess the viability of using HOST\_ID TCP options at large scale. In particular, this document provides experimentation results related to the support of the HOST\_ID TCP Options, the behavior of legacy TCP servers when receiving the HOST\_ID TCP options. This draft also discusses the impact of using a HOST\_ID TCP options on the time it takes to establish a connection; it also tries to evaluate the impact of the new TCP options on the performance of the CGN. Finally it presents the enforcement policies that could be applied by remote servers based upon the HOST\_ID options contents.

## 2. Objectives

The implementation of several HOST\_ID TCP options is primarily meant to:

- o Assess the validity of the HOST\_ID TCP option approach
- o Evaluate the impact on the TCP stack to support the HOST\_ID TCP options
- o Improve filtering and logging capabilities based upon the contents of the HOST\_ID TCP option. This means the enforcement of various policies based upon the content of the HOST\_ID TCP option at the server side: Log, Deny, Accept, etc.
- o Assess the behavior of legacy TCP servers when receiving a HOST\_ID TCP option
- o Assess the success ratio of TCP communications when a HOST\_ID TCP option is received
- o Assess the impact of injecting a HOST\_ID TCP option on the time it takes to establish a connection
- o Assess the performance impact on the CGN device that has been configured to inject the HOST\_ID option

### 3. NAT Reveal TCP Options: Overview

The original idea of defining a TCP option is documented in [I-D.wing-nat-reveal-option] and denoted as HOST\_ID\_WING.

An additional TCP option is also considered and denoted as HOST\_ID\_BOUCADAIR. The main motivation is to cover also the load-balancer use case and provide richer functionality as Forwarded-For HTTP header than HOST\_ID\_WING can provide.

The following sub-sections provide an overview of these HOST\_ID TCP options.

#### 3.1. HOST\_ID\_WING TCP Option

HOST\_ID\_WING is defined in [I-D.wing-nat-reveal-option]. Figure 1 shows the format of this option.

```

+-----+-----+-----+
|Kind=TBD|Length=4|   HOST_ID Data   |
+-----+-----+-----+
```

Figure 1: Format of HOST\_ID\_WING TCP Option

This option must be sent only upon the initial connection request, i.e., in SYN packets as shown in Figure 2

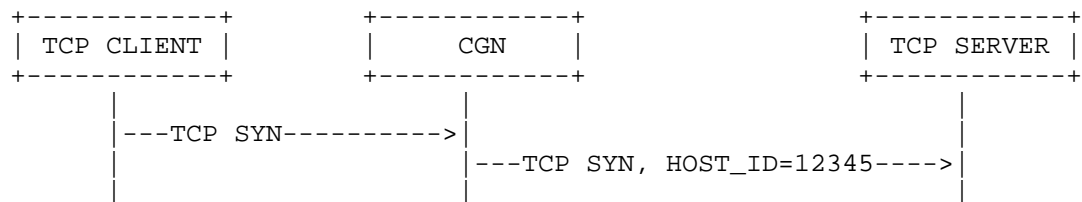


Figure 2: HOST\_ID\_WING TCP Option: Flow example

#### 3.2. HOST\_ID\_BOUCADAIR TCP Option

As mentioned above, the HOST\_ID\_BOUCADAIR TCP Option is inspired from HOST\_ID\_WING and XFF.

The HOST\_ID\_BOUCADAIR option is a 10-byte long TCP option, where KIND, Length and lifetime-Origin fields fill one byte each, and HOST\_ID data is 7-byte long as shown in Figure 3

```

+-----+-----+---+---+-----+..-----+
|Kind=TBD|Length=10| L | O |  HOST_ID_data  | HOST_ID
+-----+-----+---+---+-----+..-----+

```

Figure 3: Format of HOST\_ID\_BOUCADAIR TCP option

- o L: Indicates the validity lifetime of the enclosed data (in the spirit of [RFC6250]). The following values are supported:
  - 0: Permanent;
  - >0:Dynamic; this value indicates the validity time.
- o Origin: Indicates the origin of the data conveyed in the data field. The following values are supported:
  - 0: Internal Port
  - 1: Internal IPv4 address
  - 2: Internal Port: Internal IPv4 address
  - 3: IPv6 Prefix
  - >3: No particular semantic
- o HOST\_ID\_data depends on the content of the Origin field; padding is required.

Two modes are described below: the SYN mode (Section 3.2.1) and the ACK mode. (Section 3.2.2).

If the ACK mode is used (Section 3.2.2), Figure 4 shows the HOST\_ID\_ENABLED option (2-bytes long) to be included in the SYN.

```

+-----+-----+
|Kind=TBD|Length=2 |  HOST_ID_ENABLED
+-----+-----+

```

Figure 4: Format of HOST\_ID\_ENABLED

### 3.2.1. SYN Mode

This mode is similar to the mode described in Section 3.1. In this mode, HOST\_ID\_BOUCADAIR is sent in SYN packets.

```

+-----+-----+-----+
| TCP CLIENT |      CGN      | TCP SERVER |
+-----+-----+-----+
|
| ---TCP SYN-----> | --TCP SYN, HOST_ID=2001:db8::/5482-> |
|

```

Figure 5: HOST\_ID\_BOUCADAIR: SYN Mode

### 3.2.2. ACK Mode

The ACK Mode is as follows (see Figure 6):

- o Send HOST\_ID\_ENABLED (Figure 4) in SYN
- o If the remote TCP server supports that option, it must return it in SYNACK
- o Then the TCP Client sends an ACK in which the CGN injects HOST\_ID\_BOUCADAIR (Figure 3)

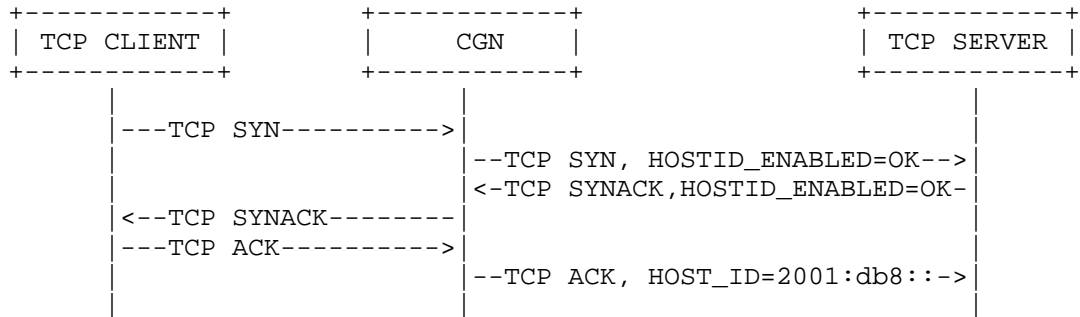


Figure 6: HOST\_ID\_BOUCADAIR: ACK Mode

## 4. Overview of the Linux Kernel Modifications

The objective of this phase is to support HOST\_ID\_WING, HOST\_ID\_BOUCADAIR and HOST\_ID\_ENABLED in the SYN mode.

In order to support the injection of the HOST\_ID TCP options presented in Section 3, some modifications were applied to the Linux Kernel (more precisely to the TCP stack part of the Kernel). The header file tcp.h, file where are defined the TCP variables and functions, is updated to define the new HOST\_ID options' KINDs (option numbers) and Lengths.

Major modifications have been made in the "tcp\_output.c" file. This file is responsible for building and transmitting all TCP packets. For each HOST\_ID TCP option, the required modifications to increase the header size and to inject KIND, Length and the corresponding HOST\_ID data are implemented for the TCP SYN packets.

As we have three different HOST\_ID options and as HOST\_ID\_BOUCADAIR can convey different information the configuration of the HOST\_ID options have to be simple with minimal complexity. Since the manipulation of HOST\_ID options impacts the Kernel TCP drivers, a suitable solution is to define new sysctl variables (system control variables) that allow the modification of Kernel parameters at

runtime, without having to reboot the machine so that it takes into account a new configuration.

Once modifications have taken place, the Kernel must be recompiled so that the new TCP options are taken into account.

Kernel modifications and recompilation have been done and tested successfully on Fedora and Debian Linux distributions, on different kernel versions.

The following configuration options are supported:

- o Enable/Disable injecting the TCP Option
- o Support HOST\_ID WING, HOST\_ID BOUCADAIR and HOST\_ID\_ENABLED
- o When the HOST\_ID TCP option is supported, the information to be injected is configurable:
  - \* Source IPv6 address or the first 56 bits of the address
  - \* Source IPv4 address
  - \* Source port number
  - \* Source IPv4 address and Source port
  - \* IPv6 address or the first 56 bits of the B4 when DS-Lite is activated

## 5. Testbed Setup & Configuration

The setup of three testbed configurations have been considered:

1. HOST\_ID TCP option is injected by the host itself. No CGN is present in the forwarding path (Figure 7)
2. HOST\_ID TCP option is injected by hosts deployed behind a HTTP proxy. No CGN is present in the forwarding path (Figure 8)
3. HOST\_ID TCP option is injected by the DS-Lite AFTR element (Figure 9).

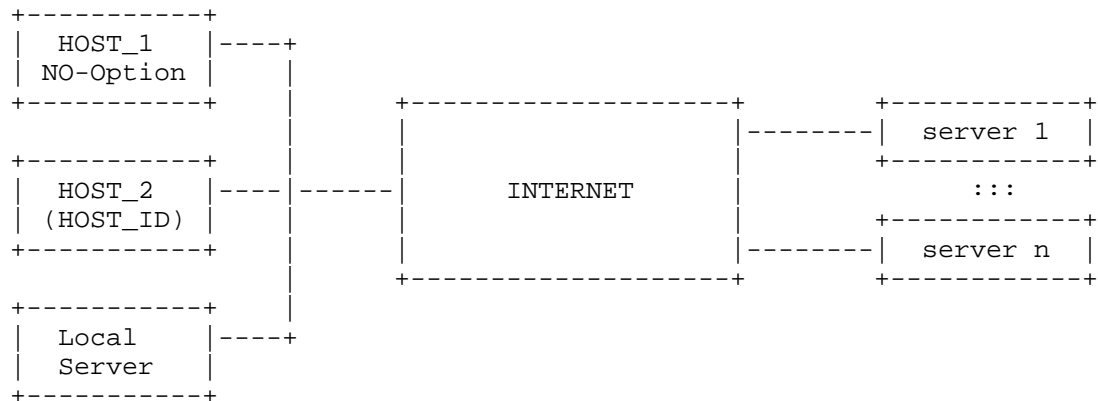


Figure 7: Testbed setup: No Proxy and no CGN

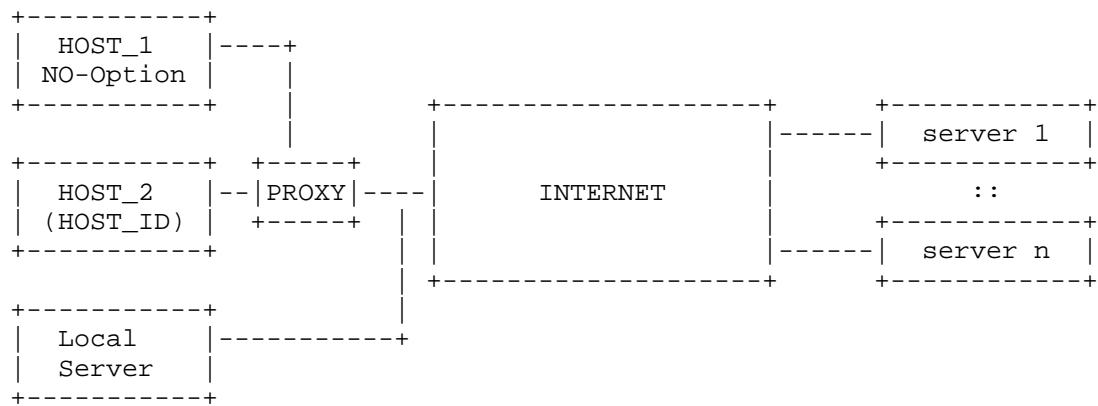


Figure 8: Testbed setup: HTTP Proxy

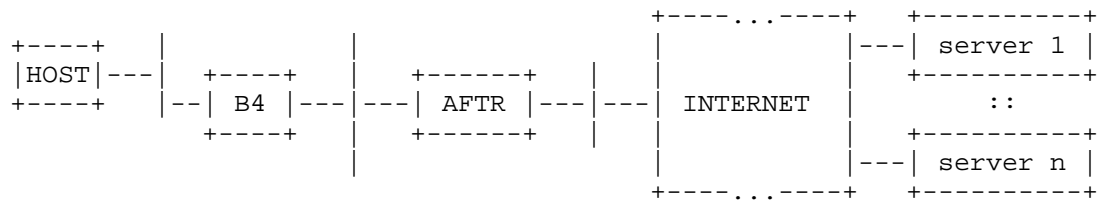


Figure 9: DS-Lite CGN Environment

Figure 7 and Figure 8 are used to assess the behavior of the top 100,000 sites when a HOST\_ID option is enabled and to evaluate the impact of the option on both the session establishment delay and the success ratio.

On the other hand, the configuration shown in Figure 9 will be used to evaluate the impact on the CGN performances when HOST\_ID TCP option is injected by the CGN.

### 5.1. Automated TCP Traffic Generator

A Python-encoded robot has been used as the traffic generator. The robot automates the retrieval of HTTP pages identified by URLs, and returns different connection information. The retrieval of pages is based upon Pycurl, a Python interface of libcurl. Libcurl is an URL transfer library that supports different protocols (e.g., HTTP, FTP).

The robot consists of two programs:

1. The first one takes an URL as a input parameter, performs the DNS lookup and then tries to connect to the corresponding machine. It returns either different time values and connection status or an error message with the source of the error in case of connection failure (e.g., DNS error). The TCP connection establishment time is calculated as the difference between the CONNECT\_TIME and NAMELOOKUP\_TIME where:
  - \* NAMELOOKUP\_TIME is the time it took from the start until the name resolution is completed.
  - \* CONNECT\_TIME is the time it took from the start until the connection to the remote host (or proxy) is completed.
2. The second program aims to increase efficiency and speed of the testing by using a multi-thread technique. It takes the number of threads and an input file listing URLs as parameters. This program prints URLs to an output file with the corresponding connection time. If something wrong happened so that the connection failed, the program returns an error message with the corresponding error type.

### 5.2. Testing Methodology and Procedure

The testing is done using two machines, one that supports the HOST\_ID TCP options and the other that does not. The second machine is used as a reference for the measurements. Testing is performed in parallel on the two machines that are directly connected to the Internet. For each HOST\_ID TCP option, the test is repeated many times. The cycle is repeated in different days. Then results are grouped into tables where averages are calculated. The comparison between the different HOST\_ID options results is made by using the no-option testing results as a reference.

Testing was also performed behind a proxy (Figure 8) to evaluate the impact of embedding the HOST\_ID TCP options on the connection establishment time when a proxy is in the path. When a proxy is

present, the connection delay is impacted (the delay is calculated for the connection between the host and the proxy).

Tests have been conducted from hosts:

1. Connected to an enterprise network
2. In a lab behind a firewall
3. Connected to two (2) commercial ISP networks

### 5.3. Check HOST\_ID TCP Options are Correctly Injected

To check whether the HOST\_ID TCP options are correctly injected, the local server in Figure 7 is configured to be reachable from Internet. Packets conveying the HOST\_ID TCP options are sent from a host supporting the options. These packets are used without alteration by the local server.

This configuration confirms the packets sent to remote servers conveys HOST\_ID TCP options.

### 5.4. Top Site List

The Alexa top sites list has been used to conduct the HTTP tests.

Anonymous FTP sites list from ftp-sites.org has been used to conduct the FTP tests.

## 6. Experimentation Results

Various combinations of the HOST\_ID TCP options have been tested:

1. HOST\_ID\_WING
2. HOST\_ID\_WING has also been adapted to include 32 bits and 64 bits values. No particular impact on session establishment has been observed.
3. HOST\_ID\_BOUCADAIR (source port)
4. HOST\_ID\_BOUCADAIR (IPv4 address)
5. HOST\_ID\_BOUCADAIR (source port:IPv4 address)
6. HOST\_ID\_BOUCADAIR (IPv6 Prefix)
7. HOST\_ID\_ENABLED

Both the success ratio and the average time to establish the TCP session are reported below.

### 6.1. HTTP Experimentation Results

Tests have been conducted from hosts:

1. Connected to an enterprise network
2. Connected to two commercial ISP networks
3. In a lab behind a firewall

#### 6.1.1.1. Configuration 1: Connected to an enterprise network

The results show that the success ratio for establishing TCP connection with legacy servers is almost the same for all the HOST\_ID options as shown in Figure 10, Figure 11 and Figure 12.

##### 6.1.1.1.1. Results

	NO-OPTION	O-WING	Failure Ratio
Top10	100,00000%	100,00000%	0,00000%
Top100	100,00000%	100,00000%	0,00000%
Top200	100,00000%	100,00000%	0,00000%
Top300	99,66667%	99,66667%	0,00000%
Top400	99,50000%	99,50000%	0,00000%
Top500	99,40000%	99,40000%	0,00000%
Top600	99,50000%	99,50000%	0,00000%
Top700	99,57143%	99,57143%	0,00000%
Top800	99,50000%	99,50000%	0,00000%
Top900	99,44444%	99,44444%	0,00000%
Top1000	99,50000%	99,50000%	0,00000%
Top2000	99,35000%	99,30000%	0,05000%
Top3000	99,10000%	99,06667%	0,03333%
Top4000	99,10000%	99,05000%	0,05000%
Top5000	99,14000%	99,10000%	0,04000%
Top6000	99,21667%	99,18333%	0,03333%
Top7000	99,25714%	99,21429%	0,04286%
Top8000	99,15000%	99,10000%	0,05000%
Top9000	99,16667%	99,12222%	0,04444%
Top10000	99,16000%	99,12000%	0,04000%
Top20000	98,50500%	98,44000%	0,06500%
Top30000	98,21667%	98,11667%	0,10000%
Top40000	98,10750%	98,00750%	0,10000%
Top50000	98,00000%	97,89800%	0,10200%
Top60000	97,95167%	97,85000%	0,10167%
Top70000	97,88857%	97,78857%	0,10000%
Top80000	97,84500%	97,74875%	0,09625%
Top90000	97,79444%	97,69889%	0,09556%
Top100000	97,75100%	97,64800%	0,10300%

Figure 10: Cumulated Success ratio (HOST\_ID\_WING)

	NO-OPTION	O-WING	Failure Ratio
1-100	100,00%	100,00%	0,00%
101-200	100,00%	100,00%	0,00%
201-300	99,00%	99,00%	0,00%
301-400	99,00%	99,00%	0,00%
401-500	99,00%	99,00%	0,00%
501-600	100,00%	100,00%	0,00%
601-700	100,00%	100,00%	0,00%
701-800	99,00%	99,00%	0,00%
801-900	99,00%	99,00%	0,00%
901-1000	100,00%	100,00%	0,00%
1-1000	99,50%	99,50%	0,00%
1001-2000	99,20%	99,10%	0,10%
2001-3000	98,60%	98,60%	0,00%
3001-4000	99,10%	99,00%	0,10%
4001-5000	99,30%	99,30%	0,00%
5001-6000	99,60%	99,60%	0,00%
6001-7000	99,50%	99,40%	0,10%
7001-8000	98,40%	98,30%	0,10%
8001-9000	99,30%	99,30%	0,00%
9001-10000	99,10%	99,10%	0,00%
10001-20000	97,85%	97,76%	0,90%
20001-30000	97,64%	97,47%	1,70%
30001-40000	97,78%	97,68%	1,00%
40001-50000	97,57%	97,46%	1,10%
50001-60000	97,71%	97,61%	1,00%
60001-70000	97,61%	97,52%	0,90%
70001-80000	97,44%	97,37%	0,70%
80001-90000	97,39%	97,30%	0,90%
90001-100000	97,36%	97,19%	1,70%

Figure 11: TopX000 Success Ratio (HOST\_ID\_WING)

	NO-OPTION	O-BOUCADAIR	Failure Ratio
1-100	100,00%	100,00%	0,00%
101-200	100,00%	100,00%	0,00%
201-300	99,00%	99,00%	0,00%
301-400	99,00%	99,00%	0,00%
401-500	99,00%	99,00%	0,00%
501-600	100,00%	100,00%	0,00%
601-700	100,00%	100,00%	0,00%
701-800	99,00%	99,00%	0,00%
801-900	99,00%	99,00%	0,00%
901-1000	100,00%	100,00%	0,00%
0-1000	99,50%	99,50%	0,00%
1001-2000	99,20%	99,10%	0,10%
2001-3000	98,60%	98,60%	0,00%
3001-4000	99,30%	99,30%	0,00%
5001-6000	99,60%	99,60%	0,00%
6001-7000	99,50%	99,40%	0,10%
7001-8000	98,40%	98,30%	0,10%
8001-9000	99,30%	99,20%	0,10%
9001-10000	99,10%	99,10%	0,00%
10001-20000	97,85%	97,76%	0,90%
20001-30000	97,64%	97,46%	1,80%
30001-40000	97,78%	97,66%	1,20%
40001-50000	97,57%	97,46%	1,10%
50001-60000	97,71%	97,61%	1,00%
60001-70000	97,61%	97,51%	1,00%
70001-80000	97,44%	97,36%	0,80%
80001-90000	97,39%	97,30%	0,90%
90001-100000	97,36%	97,19%	1,70%

Figure 12: TopX000 Success Ratio (HOST\_ID\_BOUCADAIR)

## 6.1.1.2. Analysis

- o For the top 100,000 sites, connection failures occur for 2249 HTTP sites. These failures were reported as being caused by DNS issues (servers not mounted), connection timeouts (servers down...), connection resets by peers, connection problems and empty replies from servers. The 2249 failures occur, whether HOST\_ID options are injected or not.
- o When any HOST\_ID TCP option is conveyed, 103 servers did not respond; however when no option is injected, all these servers responded normally.

- o Same results were obtained for HOST\_ID\_WING and HOST\_ID\_ENABLED.
- o Same results were obtained for all the HOST\_ID\_BOUCADAIR options (source port, IPv6 prefix, etc.).

When HOST\_ID\_BOUCADAIR is enabled, six (6) additional servers did not respond:

- o Three (3) servers (www.teufel.de - www.1001fonts.com - www.sigurros.co.uk) did not respond to the SYN packets sent by the host.
- o Three (3) servers (www.lawyers.com, www.lexis.com, www.nexis.com) responded with "strange" SYN/ACK packets with same TCP options length including a part of the HOST\_ID options that was sent. This part of HOST\_ID option caused an erroneous SYN/ACK packet received by the host: in fact the second byte of the HOST\_ID part is considered as its length and this length does not really fit with the real length of the part. So the machine does not respond back to the server with an ACK packet. This is why we have no response for these servers.

When HOST\_ID\_WING or HOST\_ID\_ENABLED is enabled, also strange SYN/ACKs were received by the host but no errors in these packets (a long series of NOP options). This justifies the connection success for these 2 options.

The results show that including a HOST\_ID TCP option does not systematically imply an extra delay for the establishment of the TCP session. Based on the average of session establishment with the top 100 000 sites, the following results have been obtained:

- o delay(HOST\_ID\_WING) < delay(NO\_OPTION): 42,55 %
- o delay(HOST\_ID\_BOUCADAIR ) < delay(NO\_OPTION): 48,16 %
- o delay(HOST\_ID\_ENABLED) < delay(NO\_OPTION): 51,28 %

#### 6.1.2. Configuration 2: In a lab behind a firewall

When a HTTP proxy is in the path, the injection of HOST\_ID TCP option does not impact the success ratio. This is due to that the HTTP proxy strips the HOST\_ID TCP options; these options are not leaked to remote Internet servers. The testing has been done by observing packets received to a server installed with a public IP address (no HOST\_ID options were seen in the received SYN packets).

#### 6.1.3. Configuration 3: Connected to two commercial ISP networks

The results obtained when testing was performed by connecting to two ISP networks confirmed the results obtained in the testing described in Section 6.1.1

## 6.1.4. Additional Results

In one of our testing for top 1000 sites, when padding was badly implemented for HOST\_ID\_BOUCADAIR (padding was implemented as a prefix so option's Length does not correspond to the real length because the padding was not counted), we got for configuration(1) in the lab and for one of the ISP the following results:

	No-Option	O-BOUCADAIR	Failure Ratio
Top10	100,00000%	100,00000%	0,00000%
Top100	100,00000%	100,00000%	0,00000%
Top200	100,00000%	100,00000%	0,00000%
Top300	100,00000%	99,66667%	0,33333%
Top400	99,75000%	99,00000%	0,75000%
Top500	99,80000%	99,00000%	0,80000%
Top600	99,83333%	98,66667%	1,16667%
Top700	99,85714%	98,14286%	1,71429%
Top800	99,75000%	98,00000%	1,75000%
Top900	99,66667%	97,33333%	2,33333%
Top1000	99,70000%	97,10000%	2,60000%

Cumulated Success ratio (HOST\_ID\_Boucadair with wrong padding)

The results for HOST\_ID\_WING for all three configurations are the same as Section 6 (this option was correctly coded). Results obtained for HOST\_ID\_BOUCADAIR are not the same.

For the configuration (2) behind a firewall, we did not face any rejection because of parsing the TCP options (the HOST\_ID options were retrieved from the packet).

## 6.1.5. Analysis

Configuration (1) in Lab and for one of the two CPEs lead to the results because 2.6% of these 1000 servers perform parsing validation for the received options so when the bad HOST\_ID\_BOUCADAIR option is sent, 2.6% of the servers treat the received SYN packets as erroneous packets and discard them.

For the connection behind the second ISP, we didn't get a response for any of the servers. After investigation, the reason was that the Box validates the received packets before sending them to the Internet. The erroneous SYN packets holding badly encoded options (HOST\_ID\_BOUCADAIR in this case) were dropped and no connection was

established. On the other hand, the other box did not validate options length for received packets before sending them to the Internet.

## 6.2. FTP

Various combinations of the HOST\_ID TCP options have been tested:

1. HOST\_ID\_WING
2. HOST\_ID\_BOUCADAIR (source port)
3. HOST\_ID\_BOUCADAIR (source port:IPv4 address)

A list of 5591 FTP servers has been used to conduct these testings. Among this list, only 2045 were reachable:

- o Failure to reach 942 FTP servers due to connection timeout
- o Failure to reach 1286 FTP servers due to DNS errors
- o Failure to reach 717 FTP servers because access was denied
- o Could not connect to 500 FTP servers
- o Response reading failed for 81 servers
- o Bad response from server for 20 servers

When HOST\_ID TCP options are injected, 9 errors are observed (connection timeout).

Figure 13 and Figure 14 provide more data about the error distribution.

	NOB	HOST_ID	Failure Ratio
1-100	100%	100%	0,000%
101-200	100%	99%	1,000%
201-300	100%	99%	1,000%
301-400	100%	100%	0,000%
401-500	100%	100%	0,000%
501-600	100%	100%	0,000%
601-700	100%	100%	0,000%
701-800	100%	100%	0,000%
801-900	100%	99%	1,000%
901-1000	100%	99%	1,000%
1001-2000	100%	99,5%	0,500%
2000-2045	100%	100%	0,000%

Figure 13: Cumulated Success Ratio (FTP)

	NOB	HOST_ID	Failure Ratio
first 10	100,000%	100,000%	0,000%
first 100	100,000%	100,000%	0,000%
first 200	100,000%	99,500%	0,500%
first 300	100,000%	99,333%	0,667%
first 400	100,000%	99,500%	0,500%
first 500	100,000%	99,600%	0,400%
first 600	100,000%	99,667%	0,333%
first 700	100,000%	99,714%	0,286%
first 800	100,000%	99,750%	0,250%
first 900	100,000%	99,667%	0,333%
first 1000	100,000%	99,600%	0,400%
first 2000	100,000%	99,550%	0,450%
first 2045	100,000%	99,560%	0,440%

Figure 14: FirstXXX FTP Servers

The results show that including a HOST\_ID TCP option does not systematically imply an extra delay for the establishment of the TCP session with remote FTP servers. Based upon the average of the session establishment with the 2045 FTP sites, the following results have been obtained:

- o delay(HOST\_ID\_WING) < delay(NO\_OPTION): 49,36585 %
- o delay(HOST\_ID\_BOUCADAIR (source port:IPv4 address)) < delay(NO\_OPTION): 48,41076%
- o delay(HOST\_ID\_BOUCADAIR (source port)) < delay(NO\_OPTION): 48,43902 %

### 6.3. SSH

The secure shell service has been tested between a host and a SSH server connected to the same network.

SSH connections have been successfully established with the server for all the HOST\_ID TCP options. Same results were obtained using configuration (1) and configuration (2).

### 6.4. Telnet

Telnet sessions have been successfully initiated for all HOST\_ID TCP options with a server (the CGN used in Figure 9).

## 7. AFTR Module Modifications

This section highlights the support the HOST\_ID functionalities in the AFTR element of the DS-Lite model (Figure 9) and presents the testing results in order to conclude about the HOST\_ID TCP options impacts on the performance of the CGN.

We used ISC AFTR implementation.

### 7.1. Specification

All privately-addressed IPv4 packets sent from DS-Lite serviced hosts go through an AFTR device where an `isc_aftr` daemon program is responsible for establishing the tunnel, configuring network interfaces and processing received packets.

The `aftr.c` source code controls all functionalities to be included or modified on packets received by the CGN, e.g., patching TCP MSS values, fix MTU, etc.

In order to activate/deactivate such functionalities, the corresponding parameters can be configured in a specific configuration file called "`aftr.conf`". In this file, other parameters are configured, e.g., the IPv6 addresses assigned to the tunnel endpoint and the global IPv4 address pool maintained by the CGN.

To support the injection of HOST\_ID TCP options, "`aftr.c`" must be updated to inject, retrieve or verify the HOST\_ID options depending on the HOST\_ID parameters defined in "`aftr.conf`" file. Four HOST\_ID parameters are defined in the configuration file:

1. `hostid`: to enable the injection, retrieval, matching... of HOST\_ID options
2. `hostid_wing`: to enable injection/verification of HOST\_ID\_WING - to disable injection or to remove HOST\_ID\_WING
3. `hostid_boucadair`: to enable injection/verification of HOST\_ID\_BOUCADAIR - to disable injection or to remove HOST\_ID\_BOUCADAIR
4. `hostid_enabled`: to enable or disable HOST\_ID\_ENABLED injection

`hostid`, `hostid_wing` and `hostid_enabled` can be simply enabled or disabled. `hostid_boucadair` can be disabled or enabled with the corresponding Origin as HOST\_ID data can be:

- o Source Port Number
- o Source IPv4 Address
- o Source IPv4 Address + Source Port Number

- o 56 bits of Tunnel Software IPv6 Source Address.

Based on different HOST\_ID parameters, the "aftr.c" code has been modified to control HOST\_ID options; the AFTR is able to:

- o Inject the enabled HOST\_ID TCP option if it is not already present in the packet
- o Retrieve an existing HOST\_ID TCP option if this option is not enabled
- o Check an existing HOST\_ID option's content if it is enabled; if the content's verification failed, the AFTR replaces the HOST\_ID contents with the suitable information

The implementation takes into consideration the SYN mode for all the HOST\_ID options (even for HOST\_ID\_enabled). The Support of HOST\_ID\_BOUCADAIR in the ACK mode needs implementation on the server's side and since both Enabled and Boucadair's options have been tested and no impact observed; the ACK mode should not imply any complication in implementation or impact on the performance.

## 7.2. Verification

The verification of HOST\_ID implementation in the CGN has taken place using the testbed setup shown in Figure 9. The host used in this testing is a modified Linux machine that can inject HOST\_ID options. The objective of the testing is to verify the different functionalities implemented in the AFTR. Verification has occurred using a local server where all the received packets were observed to make sure that the content of the HOST\_ID fields is consistent with the enabled option.

The testing consists in observing the SYN packets (as SYN mode is supported) sent by the host and in comparing these packets to those received by the server. Different combinations of HOST\_ID options sent by the host and HOST\_ID configured options at the CGN level have been used.

The results show that once the host sends packets without any HOST\_ID option injected, the SYN packets received by the server contain the correct option that has been enabled by the CGN (if any). Once HOST\_ID\_WING or HOST\_ID\_BOUCADAIR are injected by the host, if the hostid parameter in aftr.conf is enabled, the enabled (in "aftr.conf") HOST\_ID option will be injected if not already present, or else its content will be verified and corrected (if wrong); the other disabled option will be discarded if it has already been sent by the host.

One additional case has been tested when both Wing's and Boucadair's HOST\_ID options are sent by the host, the contents of the enabled

option are checked and corrected (if wrong), the other option is retrieved from the packet. The two options are dropped from the packet if they are both disabled.

The testing has been repeated for all the HOST\_ID options sent by the host and enabled by the CGN. Verification also occurred for HOST\_ID\_ENABLED option.

### 7.3. CGN Performance Testing

To conclude about the impact of using HOST\_ID, a commercial testing product has been used. This tool supports multiple application protocols such as HTTP and FTP for both IPv4 and IPv6 (including encapsulation). The DS-Lite model can be built directly from a port of this product: IPv4 packets are directly encapsulated in an IPv6 tunnel; the client's port emulates hosts and B4 elements at the same time. This port is directly connected to the AFTR tunnel endpoint. The AFTR's IPv4 interface is connected to the testing product server side where servers are assigned IPv4 addresses.

The testbed setup of this testing is shown in Figure 15:

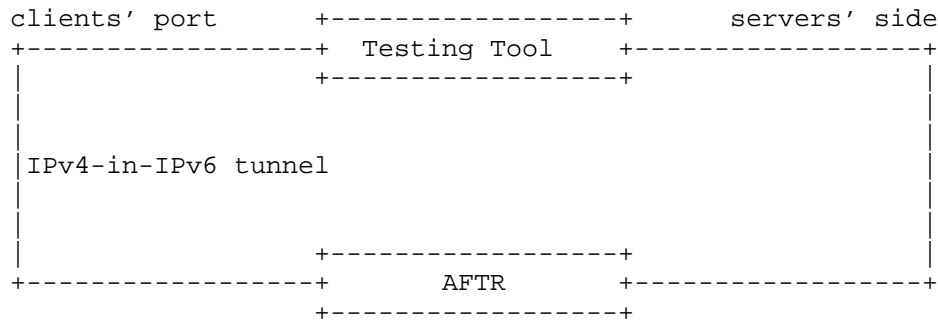


Figure 15: Platform Testbed

#### 7.3.1. Configuration

At the IP level, the testing client port was configured with IPv6 addresses representing the B4. The testing tool also supports the DS-Lite "level" where the number of clients connected to each B4 and their addresses are configured. The AFTR address is defined at this level.

In the current testing, the total number of B4 elements is 5000 behind; One client is connected to each B4 (in total, 5000 clients are configured). However, the number of active users varies from 10 to 100, 500, 1000 and 10,000 during each testing simulation.

From the server standpoint, five servers have been assigned IPv4 addresses. These servers support HTTP and FTP traffic. For each HOST\_ID TCP option, the testing was repeated for a different number of active users (N=10, 100, 500, 1000 and 10,000) and for HTTP and FTP traffic.

The HOST\_ID options are injected by the CGN.

### 7.3.2. HTTP Testing

The testing duration was about 50 seconds during which the number of active users varies as a function of time: during the first 10s, the number of active users reaches the maximum and remains the same for the next 20 s. Then it decreases to zero during the next 20s.

Hereafter are provided some testing statistics providing some details about connections' success ratio, latency and other information that can be useful to evaluate the impact of HOST\_ID on the CGN.

	No-Opt	O-WING	O-BOUCADAIR3	O-ENABLED
TCP connection established	1378	1267	1363	1369
TCP SYN SENT	1378	1267	1363	1369
Success Ratio	100	100	100	100
TCP Retries	193	193	197	177
TCP timeouts	140	136	152	111
HTTP connect' latencies t=20s	0,11	0,21	0,20	0,1
t=40s	0,40	0,50	0,50	0,45
t=60s	0,60	0,60	0,50	0,6
HTTP throughput received	46,47	45,31	45,88	46,12
TCP Connections Established/s	20,29	19,88	20,06	20,18

Figure 16: Results HTTP (N=10)

	No-Opt	O-WING	O-BOUCADAIR3	O-ENABLED
TCP connection established	1662	1739	1813	1679
TCP SYN SENT	1718	1770	1819	1729
Success Ratio	96	98	99	97
TCP Retries	1577	1569	1783	1576
TCP timeouts	798	806	934	808
HTTP connect' latencies t=20s	1,70	2,00	1,90	1,80
t=30s	3,30	2,40	2,25	3,30
t=40s	4,20	3,70	3,75	4,00
t=50s	5,00	4,80	4,50	5,00
HTTP throughput received	47,56	46,65	48,59	48,06
TCP Connections Established/s	20,94	20,53	21,35	21,19

Figure 17: Results HTTP (N=100)

	No-Opt	O-WING	O-BOUCADAIR3	O-ENABLED
TCP connection established	1956	1923	1944	1873
TCP SYN SENT	2088	2095	2137	1986
Success Ratio	93	91	90	94
TCP Retries	2734	2576	2453	2773
TCP timeouts	1261	1110	995	1213
HTTP connect' latencies t=20s	2,00	1,80	1,50	2,30
t=40s	4,00	3,30	2,80	4,30
t=50s	6,50	6,90	6,00	8,00
HTTP throughput received	70,19	65,00	69,81	67,13
TCP Connections Established/s	30,69	28,41	30,50	29,38

Figure 18: Results HTTP (N=1000)

	No-Opt	O-WING	O-BOUCADAIR4	O-ENABLED
TCP connection established	1576	2000	1796	1998
TCP SYN SENT	2088	2304	2009	2262
Success Ratio	87	86	89	88
TCP Retries	3018	3101	3013	3148
TCP timeouts	1167	1298	1213	1417
HTTP connect' latencies t=20s	2,20	3,00	2,20	2,50
t=40s	3,70	3,00	3,30	3,00
t=60s	7,80	5,00	7,00	5,60
t=70s	9,60	6,00	8,70	7,00
HTTP throughput received	45,00	54,52	51,45	57,20
TCP Connections Established/s	19,98	24,05	22,45	25,04

Figure 19: Results HTTP (N=10000)

#### 7.3.2.1. Analysis of results

The results clearly show that there is no impact of any HOST\_ID option on session establishment success ratio, which is quite similar to the success ratio when packets do not hold options or when HOST\_ID options are not used. Also, the number of established connections does not decrease when any HOST\_ID option is injected, so the CGN performance is not impacted by the fact of adding the HOST\_ID options.

Another important factor to study is the latency that can be caused by HOST\_ID injection. As the results show, the HTTP connection latency does not increase when HOST\_ID is present if we compare the latency measured at different times for the different options.

As a result, we clearly see that the average throughput measured at servers is identical, whether HOST\_ID options are used or not (given that the number of session established is quite the same).

Another consequence is that the TCP connection establishment rate at servers is not decreasing when a HOST\_ID option is taken into account.

#### 7.3.2.2. Conclusion

The results that have been obtained show that the performance of the CGN is not impacted by HOST\_ID option injection even when the number of active users is high (10,000 is not negligible for a CGN run on an ordinary Linux machine): neither the session success ratio, nor the connection latency are impacted by the presence of the HOST\_ID in SYN

packets.

### 7.3.3. FTP

The same testing was also run for FTP traffic. No particular impact on the performance of the CGN has been observed.

## 8. IPTABLES: Modifications to Enforce Policies at the Server Side

### 8.1. Overview

iptables module has been updated to:

- o Log the content of TCP header with HOST\_ID
- o Drop packets holding a HOST\_ID option
- o Match any HOST\_ID value
- o Drop packets holding a specific HOST\_ID value
- o Strip any existing HOST\_ID option

To support the above functionalities, modification should take into consideration stripping and matching options as described below:

1. To strip the content of any existing HOST\_ID option, the shared library "libxt\_TCPOPTSTRIP.so" is modified: the HOST\_ID\_WING and HOST\_ID\_BOUCADAIR Kinds' numbers were defined in the corresponding source file (libxt\_TCPOPTSTRIP.c) with the corresponding names to enforce the iptables stripping rule. After enforcing these changes, the shared library must be created to replace the existing one and to allow applying the rule of stripping of the HOST\_ID options. Once modifications have taken place, the following command should be used to strip the HOST\_ID options:

```
iptables -t mangle -A INPUT -j TCPOPTSTRIP -p tcp --strip-options  
hostid_wing, hostid_boucadair
```

2. In order to allow blocking, logging or applying any rule based upon the HOST\_ID\_WING or HOST\_ID\_BOUCADAIR values or range of values, a HOST\_ID shared library must be created to:
  - \* Match HOST\_ID options values entered in corresponding iptables rules,
  - \* Print the HOST\_ID rules on screen,
  - \* Save values,
  - \* Check the values (or range values) entered by user if they respect the limit values of these options.

In addition to the shared library: a specific Kernel module must be built to apply HOST\_ID matching rules on the packets passing through the network interfaces. This module compares the HOST\_ID options' values held by packets with the HOST\_ID values specified in the iptables rule table: when a packet matches the HOST\_ID's range, the corresponding rule will be applied for this packet. The HOST\_ID\_WING matching value is 2 bytes long corresponding to HOST\_ID\_WING data.

The HOST\_ID\_BOUCADAIR matching value is 8 bytes long corresponding to Lifetime + Origin field (1 byte) and HOST\_ID\_WING data (7 bytes).

## 8.2. Validation

After having updated the iptables package with the suitable HOST\_ID libraries and module, different HOST\_ID policies should be applied and tested on the server side. The testing has been done using a simple configuration as shown below (Figure 20).

```

+-----+      +-----+      +-----+      +-----+
|  HOST  |-----|   B4   |-----|  AFTR  |-----| local server |
+-----+      +-----+      +-----+      +-----+

```

Figure 20: Platform configuration: HOST\_ID enforcing policies

In the current testing, the AFTR supports HOST\_ID options injection and iptables is modified at the local server. Logging recommendations consists of logging the IPv4 address and the HOST\_ID option for each connection. Because HOST\_ID is sent only in SYN packets (in the current implementation), only SYN packets will be logged to a specific file called iptables.log: the rsyslog.d must be updated with the corresponding command to log iptables messages into the specific file. Then rsyslog must be reloaded to apply changes.

## 8.3. Stripping HOST\_ID Options

To strip a certain HOST\_ID option, TCPOPTSTRIP rule must be called. Verification consists in logging and then checking the SYN packets and more precisely the corresponding TCP options, e.g., the following rules must be applied to strip HOST\_ID\_WING:

```

iptables -t mangle -A INPUT -j TCPOPTSTRIP -p tcp --strip-options
hostid_wing
iptables -A INPUT -j LOG --log-tcp-options -p tcp --syn

```

The first rule applies for the mangle table. This table allows

stripping HOST\_ID\_WING whose role is to remove option Wing's fields and replaces them by NOP options (NOP=No Operation=0x01). The second rule enables the logging of SYN packets with the corresponding TCP options.

After applying these rules (to strip and log HOST\_ID\_WING) on the local server, we tried to access the server's HTTP pages from the host. The test is repeated several times and a different HOST\_ID option is enabled by the AFTR each time.

Then the "iptables.log" file is checked: only one SYN packet is logged with 4 bytes stripped out in the TCP option part. All IPv4 packets going through the AFTR are also logged to be compared with the server's logged stripped packets.

The comparison of the SYN packets logged by the server with the SYN packets sent by the AFTR clearly shows that the stripped option is HOST\_ID\_WING (all the header fields have been verified to ensure packet matching): the 4 bytes corresponding to the HOST\_ID\_WING option are replaced with NOP options (each one of the 4 bytes is equal to '1' = NOP).

The same testing was repeated with HOST\_ID\_BOUCADAIR. The testing shows that the 10 bytes corresponding to this option were successfully stripped.

#### 8.4. Logging a Specific HOST\_ID Option Value

The remote server should be able to track connections coming from different clients; it should log packets headers including the HOST\_ID TCP option information. This can be enforced using the following command:

```
iptables -t mangle -A INPUT -j TCPOPTSTRIP -p tcp --strip-options  
hostid_wing
```

Now, to log packets matching a certain HOST\_ID value or range of values, the following rule must be applied:

```
iptables -A INPUT -p tcp --syn -m hostid --hostid_wing value[:value]  
-j LOG --log-tcp-options
```

This command matches the HOST\_ID\_WING values held by SYN packets with the specific value [or the specific range of values] determined by

the rule.

The testing configuration in Figure 20 was used. The HOST\_ID\_WING data are implemented as being the last 16 bits of the IPv4 private source address. When the HOST\_ID\_WING option is injected by the CGN, if the data field value corresponds to the iptables value (or range of values), the packet header is logged. Otherwise, if the HOST\_ID\_WING data is said out of range or the packet does not hold the HOST\_ID\_WING option, the packet is not logged.

The same testing was repeated to match HOST\_ID\_BOUCADAIR data information:

```
iptables -A INPUT -p tcp --syn -m hostid --hostid_boucadair value  
[:value] -j LOG -log-tcp-options
```

To verify the logging of a specific Boucadair's value, the Boucadair's options holding source IP address (Origin=2) or IPv6 prefix (Origin=4) were tested successfully; these data values are fixed since they depend on the host's address. The two other options that include source port numbers (variable) cannot be tested by value because the port number varies for each connection.

The iptables rules to log HOST\_ID\_BOUCADAIR range values have been verified successfully for all four HOST\_ID\_BOUCADAIR options.

#### 8.5. Dropping a specific HOST\_ID Option Value

The same testing methodology described in the previous section was repeated to drop packets matching HOST\_ID value (or a range of values); e.g. to drop SYN packets matching a particular HOST\_ID\_WING value:

```
iptables -A INPUT -p tcp --syn -m hostid --hostid_wing value[:value]  
-j DROP
```

In this testing, the HOST\_ID\_WING option is enabled at the CGN level. After applying the previous rule where Wing's specified value corresponds to the HOST\_ID\_WING data value (last 16 bits of the host's IPv4 source address), the hosts tries to access HTTP pages of the local server. It sends SYN packets but the server does not respond. Because this packet matches the iptables matching value, the corresponding rule is applied to the SYN packets: a SYN packet is dropped so the host does not receive any packet in return.

When the host is still trying to retrieve pages by sending SYN packets, the command 'iptables -F' will flush all iptables rules. Once applied, this command will let the host retrieve the required pages and the connection is therefore established successfully.

The same testing was repeated for HOST\_ID\_BOUCADAIR options. SYN packets matching the corresponding rule value or range of values were dropped. Once iptables rules are flushed, connection is established normally.

## 9. IANA Considerations

This document makes no request of IANA.

## 10. Security Considerations

Security considerations discussed in [I-D.wing-nat-reveal-option] should be taken into account.

## 11. Acknowledgments

Many thanks to M. Meulle, P. Ng Tung and L. Valeyre for their help and review. Special thanks to C. Jacquenet for his careful review.

## 12. References

### 12.1. Normative References

- [I-D.wing-nat-reveal-option]  
Yourtchenko, A. and D. Wing, "Revealing hosts sharing an IP address using TCP option",  
draft-wing-nat-reveal-option-03 (work in progress),  
December 2011.
- [RFC6250] Thaler, D., "Evolution of the IP Model", RFC 6250,  
May 2011.

### 12.2. Informative References

- [I-D.boucadair-intarea-nat-reveal-analysis]  
Boucadair, M., Touch, J., Levis, P., and R. Penno,  
"Analysis of Solution Candidates to Reveal a Host Identifier in Shared Address Deployments",  
draft-boucadair-intarea-nat-reveal-analysis-04 (work in progress),  
December 2011.

progress), September 2011.

[RFC6269] Ford, M., Boucadair, M., Durand, A., Levis, P., and P. Roberts, "Issues with IP Address Sharing", RFC 6269, June 2011.

#### Authors' Addresses

Elie Abdo  
France Telecom  
Issy-les-Moulineaux

Email: [elie.abdo@orange.com](mailto:elie.abdo@orange.com)

Mohamed Boucadair  
France Telecom

Email: [mohamed.boucadair@orange.com](mailto:mohamed.boucadair@orange.com)

Jaqueline Queiroz  
France Telecom  
Issy-les-Moulineaux

Email: [jaqueline.queiroz@orange.com](mailto:jaqueline.queiroz@orange.com)



TCP Maintenance Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: August 29, 2013

N. Dukkipati  
N. Cardwell  
Y. Cheng  
M. Mathis  
Google, Inc  
February 25, 2013

Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses  
draft-dukkipati-tcpm-tcp-loss-probe-01.txt

## Abstract

Retransmission timeouts are detrimental to application latency, especially for short transfers such as Web transactions where timeouts can often take longer than all of the rest of a transaction. The primary cause of retransmission timeouts are lost segments at the tail of transactions. This document describes an experimental algorithm for TCP to quickly recover lost segments at the end of transactions or when an entire window of data or acknowledgments are lost. Tail Loss Probe (TLP) is a sender-only algorithm that allows the transport to recover tail losses through fast recovery as opposed to lengthy retransmission timeouts. If a connection is not receiving any acknowledgments for a certain period of time, TLP transmits the last unacknowledged segment (loss probe). In the event of a tail loss in the original transmissions, the acknowledgment from the loss probe triggers SACK/FAACK based fast recovery. TLP effectively avoids long timeouts and thereby improves TCP performance.

## Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 29, 2013.

## Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Terminology . . . . .	5
2. Loss probe algorithm . . . . .	5
2.1. Pseudocode . . . . .	6
2.2. FACK threshold based recovery . . . . .	8
3. Detecting recovered losses . . . . .	9
3.1. TLP Loss Detection: The Basic Idea . . . . .	9
3.2. TLP Loss Detection: Algorithm Details . . . . .	9
4. Discussion . . . . .	11
4.1. Unifying loss recoveries . . . . .	12
4.2. Recovery of any N-degree tail loss . . . . .	12
5. Experiments with TLP . . . . .	14
6. Related work . . . . .	16
7. Security Considerations . . . . .	17
8. IANA Considerations . . . . .	17
9. References . . . . .	18
Authors' Addresses . . . . .	19

## 1. Introduction

Retransmission timeouts are detrimental to application latency, especially for short transfers such as Web transactions where timeouts can often take longer than all of the rest of a transaction. This document describes an experimental algorithm, Tail Loss Probe (TLP), to invoke fast recovery for losses that would otherwise be only recoverable through timeouts.

The Transmission Control Protocol (TCP) has two methods for recovering lost segments. First, the fast retransmit algorithm relies on incoming duplicate acknowledgments (ACKs), which indicate that the receiver is missing some data. After a required number of duplicate ACKs have arrived at the sender, it retransmits the first unacknowledged segment and continues with a loss recovery algorithm such as the SACK-based loss recovery [RFC6675]. If the fast retransmit algorithm fails for any reason, TCP uses a retransmission timeout as the last resort mechanism to recover lost segments. If an ACK for a given segment is not received in a certain amount of time called retransmission timeout (RTO), the segment is resent [RFC6298].

Timeouts can occur in a number of situations, such as the following:

- (1) Drop tail at the end of transactions. Example: consider a transfer of five segments sent on a connection that has a congestion window of ten. Any degree of loss in the tail, such as segments four and five, will only be recovered via a timeout.
- (2) Mid-transaction loss of an entire window of data or ACKs. Unlike (1) there is more data waiting to be sent. Example: consider a transfer of four segments to be sent on a connection that has a congestion window of two. If the sender transmits two segments and both are lost then the loss will only be recovered via a timeout.
- (3) Insufficient number of duplicate ACKs to trigger fast recovery at sender. The early retransmit mechanism [RFC5827] addresses this problem in certain special circumstances, by reducing the number of duplicate ACKs required to trigger a fast retransmission.
- (4) An unexpectedly long round-trip time (RTT), such that the ACKs arrive after the RTO timer expires. The F-RTO algorithm [RFC5682] is designed to detect such spurious retransmission timeouts and at least partially undo the consequences of such events.

Measurements on Google Web servers show that approximately 70% of retransmissions for Web transfers are sent after the RTO timer expires, while only 30% are handled by fast recovery. Even on servers exclusively serving YouTube videos, RTO based retransmissions

account for about 46% of the retransmissions. If the losses are detectable from the ACK stream (through duplicate ACKs or SACK blocks) then early retransmit, fast recovery and proportional rate reduction are effective in avoiding timeouts [IMC11PRR]. Timeout retransmissions that occur in recovery and disorder state (a state indicating that a connection has received some duplicate ACKs), account for just 4% of the timeout episodes. On the other hand 96% of the timeout episodes occur without any preceding duplicate ACKs or other indication of losses at the sender [IMC11PRR]. Early retransmit and fast recovery have no hope of repairing losses without these indications. Efficiently addressing situations that would cause timeouts without any prior indication of losses is a significant opportunity for additional improvements to loss recovery.

To get a sense of just how long the RTOs are in relation to connection RTTs, following is the distribution of RTO/RTT values on Google Web servers. [percentile, RTO/RTT]: [50th percentile, 4.3]; [75th percentile, 11.3]; [90th percentile, 28.9]; [95th percentile, 53.9]; [99th percentile, 214]. Large RTOs, typically caused by variance in measured RTTs, can be a result of intermediate queuing, and service variability in mobile channels. Such large RTOs make a huge contribution to the long tail on the latency statistics of short flows. Note that simply reducing the length of RTO does not address the latency problem for two reasons: first, it increases the chances of spurious retransmissions. Second and more importantly, an RTO reduces TCP's congestion window to one and forces a slow start. Recovery of losses without relying primarily on the RTO mechanism is beneficial for short TCP transfers.

The question we address in this document is: Can a TCP sender recover tail losses of transactions through fast recovery and thereby avoid lengthy retransmission timeouts? We specify an algorithm, Tail Loss Probe (TLP), which sends probe segments to trigger duplicate ACKs with the intent of invoking fast recovery more quickly than an RTO at the end of a transaction. TLP is applicable only for connections in Open state, wherein a sender is receiving in-sequence ACKs and has not detected any lost segments. TLP can be implemented by modifying only the TCP sender, and does not require any TCP options or changes to the receiver for its operation. For convenience, this document mostly refers to TCP, but the algorithms and other discussion are valid for Stream Control Transmission Protocol (SCTP) as well.

This document is organized as follows. Section 2 describes the basic Loss Probe algorithm. Section 3 outlines an algorithm to detect the cases when TLP plugs a hole in the sender. The algorithm makes the sender aware that a loss had occurred so it performs the appropriate congestion window reduction. Section 4 discusses the interaction of TLP with early retransmit in being able to recover any degree of tail

losses. Section 5 discusses the experimental results with TLP on Google Web servers. Section 6 discusses related work, and Section 7 discusses the security considerations.

### 1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 2. Loss probe algorithm

The Loss probe algorithm is designed for a sender to quickly detect tail losses without waiting for an RTO. We will henceforth use tail loss to generally refer to either drops at the tail end of transactions or a loss of an entire window of data/ACKs. TLP works for senders with SACK enabled and in Open state, i.e. the sender has so far received in-sequence ACKs with no SACK blocks. The risk of a sender incurring a timeout is high when the sender has not received any ACKs for a certain portion of time but is unable to transmit any further data either because it is application limited (out of new data to send), receiver window (rwnd) limited, or congestion window (cwnd) limited. For these circumstances, the basic idea of TLP is to transmit probe segments for the specific purpose of eliciting additional ACKs from the receiver. The initial idea was to send some form of zero window probe (ZWP) with one byte of new or old data. The ACK from the ZWP would provide an additional opportunity for a SACK block to detect loss without an RTO. Additional losses can be detected subsequently and repaired as SACK based fast recovery proceeds. However, in practice sending a single byte of data turned out to be problematic to implement and more fragile than necessary. Instead we use a full segment to probe but have to add complexity to compensate for the probe itself masking losses.

Define probe timeout (PTO) to be a timer event indicating that an ACK is overdue on a connection. The PTO value is set to  $\max(2 * \text{SRTT}, 10\text{ms})$ , where SRTT is the smoothed round-trip time [RFC6298], and is adjusted to account for delayed ACK timer when there is only one outstanding segment.

The basic version of the TLP algorithm transmits one probe segment after a probe timeout if the connection has outstanding unacknowledged data but is otherwise idle, i.e. not receiving any ACKs or is cwnd/rwnd/application limited. The transmitted segment, aka loss probe, can be either a new segment if available and the receive window permits, or a retransmission of the most recently sent segment, i.e., the segment with the highest sequence number. When

there is tail loss, the ACK from the probe triggers fast recovery. In the absence of loss, there is no change in the congestion control or loss recovery state of the connection, apart from any state related to TLP itself.

TLP MUST NOT be used for non-SACK connections. SACK feedback allows senders to use the algorithm described in section 3 to infer whether any segments were lost.

## 2.1. Pseudocode

We define the terminology used in specifying the TLP algorithm:

FlightSize: amount of outstanding data in the network as defined in [RFC5681].

RTO: The transport's retransmission timeout (RTO) is based on measured round-trip times (RTT) between the sender and receiver, as specified in [RFC6298] for TCP.

PTO: Probe timeout is a timer event indicating that an ACK is overdue. Its value is constrained to be smaller than or equal to an RTO.

SRTT: smoothed round-trip time computed like in [RFC6298].

Open state: the sender has so far received in-sequence ACKs with no SACK blocks, and no other indications (such as retransmission timeout) that a loss may have occurred.

Consecutive PTOs: back-to-back PTOs all scheduled for the same tail packets in a flight. The (N+1)st PTO is scheduled after transmitting the probe segment for Nth PTO.

The TLP algorithm works as follows:

(1) Schedule PTO after transmission of new data in Open state:

Check for conditions to schedule PTO outlined in step 2 below.

FlightSize > 1: schedule PTO in  $\max(2 \cdot \text{SRTT}, 10\text{ms})$ .

FlightSize == 1: schedule PTO in  $\max(2 \cdot \text{SRTT}, 1.5 \cdot \text{SRTT} + \text{WCDelAckT})$ .

If RTO is earlier, schedule PTO in its place:  $\text{PTO} = \min(\text{RTO}, \text{PTO})$ .

WCDelAckT stands for worst case delayed ACK timer. When FlightSize is 1, PTO is inflated additionally by WCDelAckT time to compensate for a potential long delayed ACK timer at the receiver. The RECOMMENDED value for WCDelAckT is 200ms.

A PTO value of  $2 \times \text{SRTT}$  allows a sender to wait long enough to know that an ACK is overdue. Under normal circumstances, i.e. no losses, an ACK typically arrives in one RTT. But choosing PTO to be exactly an RTT is likely to generate spurious probes given that even end-system timings can easily push an ACK to be above an RTT. We chose PTO to be the next integral value of RTT. If RTO is smaller than the computed value for PTO, then a probe is scheduled to be sent at the RTO time. The RTO timer is rearmed at the time of sending the probe, as is shown in Step 3 below. This ensures that a PTO is always sent prior to a connection experiencing an RTO.

(2) Conditions for scheduling PTO:

- (a) Connection is in Open state.
- (b) Connection is either cwnd limited or application limited.
- (c) Number of consecutive PTOs  $\leq 2$ .
- (d) Connection is SACK enabled.

Implementations MAY use one or two consecutive PTOs.

(3) When PTO fires:

- (a) If a new previously unsent segment exists:
    - > Transmit new segment.
    - > FlightSize += MSS. cwnd remains unchanged.
  - (b) If no new segment exists:
    - > Retransmit the last segment.
  - (c) Increment statistics counter for loss probes.
  - (d) If conditions in (2) are satisfied:
    - > Reschedule next PTO.
- Else:
- > Rearm RTO to fire at epoch 'now+RTO'.

The reason for retransmitting the last segment in Step (b) is so that the ACK will carry SACK blocks and trigger either SACK-based loss recovery [RFC6675] or FACK threshold based fast recovery [FACK]. On transmission of a TLP, a MIB counter is incremented to keep track of the total number of loss probes sent.

(4) During ACK processing:

Cancel any existing PTO.

If conditions in (2) allow:

- > Reschedule PTO relative to the ACK receipt time.

Following is an example of TLP. All events listed are at a TCP sender.

(1) Sender transmits segments 1-10: 1, 2, 3, ..., 8, 9, 10. There is no more new data to transmit. A PTO is scheduled to fire in 2 RTTs, after the transmission of the 10th segment.

(2) Receives acknowledgements (ACKs) for segments 1-5; segments 6-10 are lost and no ACKs are received. Note that the sender (re)schedules its PTO timer relative to the last received ACK, which is the ACK for segment 5 in this case. The sender sets the PTO interval using the calculation described in step (1) of the algorithm.

(3) When PTO fires, sender retransmits segment 10.

(4) After an RTT, SACK for packet 10 arrives. The ACK also carries SACK holes for segments 6, 7, 8 and 9. This triggers FACK threshold based recovery.

(5) Connection enters fast recovery and retransmits remaining lost segments.

## 2.2. FACK threshold based recovery

At the core of TLP is its reliance on FACK threshold based algorithm to invoke Fast Recovery. In this section we specify this algorithm.

Section 3.1 of the Forward Acknowledgement (FACK) Paper [FACK] describes an alternate algorithm for triggering fast retransmit, based on the extent of the SACK scoreboard. Its goal is to trigger fast retransmit as soon as the receiver's reassembly queue is larger than the dupack threshold, as indicated by the difference between the forward most SACK block edge and SND.UNA. This algorithm quickly and reliably triggers fast retransmit in the presence of burst losses -- often on the first SACK following such a loss. Such a threshold based algorithm also triggers fast retransmit immediately in the presence of any reordering with extent greater than the dupack threshold.

FACK threshold based recovery works by introducing a new TCP state variable at the sender called SND.FACK. SND.FACK reflects the forward-most data held by the receiver and is updated when a SACK block is received acknowledging data with a higher sequence number than the current value of SND.FACK. SND.FACK reflects the highest sequence number known to have been received plus one. Note that in non-recovery states, SND.FACK is the same as SND.UNA. The following snippet is the pseudocode for FACK threshold based recovery.

```
If (SND.FACK - SND.UNA) > dupack threshold:  
-> Invoke Fast Retransmit and Fast Recovery.
```

### 3. Detecting recovered losses

If the only loss was the last segment, there is the risk that the loss probe itself might repair the loss, effectively masking it from congestion control. To avoid interfering with mandatory congestion control [RFC5681] it is imperative that TLP include a mechanism to detect when the probe might have masked a loss and to properly reduce the congestion window (cwnd). An algorithm to examine subsequent ACKs to determine whether the original segment was lost is described here.

Since it is observed that a significant fraction of the hosts that support SACK do not support duplicate selective acknowledgments (D-SACKs) [RFC2883] the TLP algorithm for detecting such lost segments relies only on basic RFC 2018 SACK [RFC2018].

#### 3.1. TLP Loss Detection: The Basic Idea

Consider a TLP retransmission "episode" where a sender retransmits N consecutive TLP packets, all for the same tail packet in a flight. Let us say that an episode ends when the sender receives an ACK above the SND.NXT at the time of the episode. We want to make sure that before the episode ends the sender receives N "TLP dupacks", indicating that all N TLP probe segments were unnecessary, so there was no loss/hole that needed plugging. If the sender gets less than N "TLP dupacks" before the end of the episode, then probably the first TLP packet to arrive at the receiver plugged a hole, and only the remaining TLP packets that arrived at the receiver generated dupacks.

Note that delayed ACKs complicate the picture, since a delayed ACK will imply that the sender receives one fewer ACK than would normally be expected. To mitigate this complication, before sending a TLP loss probe retransmission, the sender should attempt to wait long enough that the receiver has sent any delayed ACKs that it is withholding. The sender algorithm, described in section 2.1 features such a delay.

If there is ACK loss or a delayed ACK, then this algorithm is conservative, because the sender will reduce cwnd when in fact there was no packet loss. In practice this is acceptable, and potentially even desirable: if there is reverse path congestion then reducing cwnd is prudent.

#### 3.2. TLP Loss Detection: Algorithm Details

##### (1) State

TLPRtxOut: the number of unacknowledged TLP retransmissions in current TLP episode. The connection maintains this integer counter that tracks the number of TLP retransmissions in the current episode for which we have not yet received a "TLP dupack". The sender initializes the TLPRtxOut field to 0.

TLPHighRxt: the value of SND.NXT at the time of TLP retransmission. The TLP sender uses TLPHighRxt to record SND.NXT at the time it starts doing TLP transmissions during a given TLP episode.

## (2) Initialization

When a connection enters the ESTABLISHED state, or suffers a retransmission timeout, or enters fast recovery, it executes the following:

```
TLPRtxOut = 0;
TLPHighRxt = 0;
```

## (3) Upon sending a TLP retransmission:

```
if (TLPRtxOut == 0)
    TLPHighRxt = SND.NXT;
TLPRtxOut++;
```

## (4) Upon receiving an ACK:

### (a) Tracking ACKs

We define a "TLP dupack" as a dupack that has all the regular properties of a dupack that can trigger fast retransmit, plus the ACK acknowledges TLPHighRxt, and the ACK carries no new SACK information (as noted earlier, TLP requires that the receiver supports SACK). This is the kind of ACK we expect to see for a TLP transmission if there were no losses. More precisely, the TLP sender considers a TLP probe segment as acknowledged if all of the following conditions are met:

- (a) TLPRtxOut > 0
- (b) SEG.ACK == TLPHighRxt
- (c) the segment contains no SACK blocks for sequence ranges above TLPHighRxt
- (d) the ACK does not advance SND.UNA
- (e) the segment contains no data
- (f) the segment is not a window update

If all of those conditions are met, then the sender executes the following:

```
TLPRtxOut--;
```

(b) Marking the end of a TLP episode and detecting losses

If an incoming ACK is after `TLPHighRxt`, then the sender deems the TLP episode over. At that time, the TLP sender executes the following:

```
isLoss = (TLPRtxOut > 0) &&  
         (segment does not carry a DSACK for TLP retransmission);  
TLPRtxOut = 0  
if (isLoss)  
    EnterRecovery();
```

In other words, if the sender detects an ACK for data beyond the TLP loss probe retransmission then (in the absence of reordering on the return path of ACKs) it should have received any ACKs that will indicate whether the original or any loss probe retransmissions were lost. An exception is the case when the segment carries a Duplicate SACK (DSACK) for the TLP retransmission. If the `TLPRtxOut` count is still non-zero and thus indicates that some TLP probe segments remain unacknowledged, then the sender should presume that at least one segment was lost, so it should enter fast recovery using the proportional rate reduction algorithm [IMC11PRR].

(5) Senders must only send a TLP loss probe retransmission if all the conditions from section 2.1 are met and the following condition also holds:

```
(TLPRtxOut == 0) || (SND.NXT == TLPHighRxt)
```

This ensures that there is at most one sequence range with outstanding TLP retransmissions. The sender maintains this invariant so that there is at most one TLP retransmission "episode" happening at a time, so that the sender can use the algorithm described above in this section to determine when the episode is over, and thus when it can infer whether any data segments were lost.

Note that this condition only limits the number of outstanding TLP loss probes that are retransmissions. There may be an arbitrary number of outstanding unacknowledged TLP loss probes that consist of new, previously-unsent data, since the standard retransmission timeout and fast recovery algorithms are sufficient to detect losses of such probe segments.

#### 4. Discussion

In this section we discuss two properties related to TLP.

#### 4.1. Unifying loss recoveries

The existing loss recovery algorithms in TCP have a discontinuity: A single segment loss in the middle of a packet train can be recovered via fast recovery while a loss at the end of the train causes an RTO. Example: consider a train of segments 1-10, loss of segment five can be recovered quickly through fast recovery, while loss of segment ten can only be recovered through a timeout. In practice, the difference between losses that trigger RTO versus those invoking fast recovery has more to do with the position of the losses as opposed to the intensity or magnitude of congestion at the link.

TLP unifies the loss recovery mechanisms regardless of the position of a loss, so now with TLP a segment loss in the middle of a train as well as at the tail end can now trigger the same fast recovery mechanisms.

#### 4.2. Recovery of any N-degree tail loss

The TLP algorithm, when combined with a variant of the early retransmit mechanism described below, is capable of recovering any tail loss for any sized flow using fast recovery.

We propose the following enhancement to the early retransmit algorithm described in [RFC5827]: in addition to allowing an early retransmit in the scenarios described in [RFC5827], we propose to allow a delayed early retransmit [IMC11PRR] in the case where there are three outstanding segments that have not been cumulatively acknowledged and one segment that has been fully SACKed.

Consider the following scenario, which illustrates an example of how this enhancement allows quick loss recovery in a new scenario:

- (1) scoreboard reads: A \_ \_ \_
- (2) TLP retransmission probe of the last (fourth) segment
- (3) the arrival of a SACK for the last segment changes scoreboard to: A \_ \_ S
- (4) early retransmit and fast recovery of the second and third segments

With this enhancement to the early retransmit mechanism, then for any degree of N-segment tail loss we get a quick recovery mechanism instead of an RTO.

Consider the following taxonomy of tail loss scenarios, and the ultimate outcome in each case:

	number of losses	scoreboard after TLP retrans ACKed	mechanism	final outcome
(1)	AAAL	AAAA	TLP loss detection	all repaired
(2)	AALL	AALS	early retransmit	all repaired
(3)	ALLL	ALLS	early retransmit	all repaired
(4)	LLLL	LLLS	FAK fast recovery	all repaired
(5)	>=5 L	..LS	FAK fast recovery	all repaired

key:

A = ACKed segment

L = lost segment

S = SACKed segment

Let us consider each tail loss scenario in more detail:

(1) With one segment lost, the TLP loss probe itself will repair the loss. In this case, the sender's TLP loss detection algorithm will notice that a segment was lost and repaired, and reduce its congestion window in response to the loss.

(2) With two segments lost, the TLP loss probe itself is not enough to repair the loss. However, when the SACK for the loss probe arrives at the sender, then the early retransmit mechanism described in [RFC5827] will note that with two segments outstanding and the second one SACKed, the sender should retransmit the first segment. This retransmit will repair the single remaining lost segment.

(3) With three segments lost, the TLP loss probe itself is not enough to repair the loss. However, when the SACK for the loss probe arrives at the sender, then the enhanced early retransmit mechanism described in this section will note that with three segments outstanding and the third one SACKed, the sender should retransmit the first segment and enter fast recovery. The early retransmit and fast recovery phase will, together, repair the the remaining two lost segments.

(4) With four segments lost, the TLP loss probe itself is not enough to repair the loss. However, when the SACK for the loss probe arrives at the sender, then the FACK fast retransmit mechanism [FACK] will note that with four segments outstanding and the fourth one SACKed, the sender should retransmit the first segment and enter fast recovery. The fast retransmit and fast recovery phase will, together, repair the the remaining two lost segments.

(5) With five or more segments lost, events precede much as in case (4). The TLP loss probe itself is not enough to repair the loss.

However, when the SACK for the loss probe arrives at the sender, then the FACK fast retransmit mechanism [FACK] will note that with five or more segments outstanding and the segment highest in sequence space SACKed, the sender should retransmit the first segment and enter fast recovery. The fast retransmit and fast recovery phase will, together, repair the remaining lost segments.

In summary, the TLP mechanism, in conjunction with the proposed enhancement to the early retransmit mechanism, is able to recover from a tail loss of any number of segments without resort to a costly RTO.

## 5. Experiments with TLP

In this section we describe experiments and measurements with TLP performed on Google Web servers using Linux 2.6. The experiments were performed over several weeks and measurements were taken across a wide range of Google applications. The main goal of the experiments is to instrument and measure TLP's performance relative to the baseline. The experiment and baseline were using the same kernels with an on/off switch to enable TLP.

Our experiments include both the basic TLP algorithm of Section 2 and its loss detection component in Section 3. All other algorithms such as early retransmit and FACK threshold based recovery are present in the both the experiment and baseline. There are three primary metrics we are interested in: impact on TCP latency (average and tail or 99th percentile latency), retransmission statistics, and the overhead of probe segments relative to the total number of transmitted segments. TCP latency is the time elapsed between the server transmitting the first byte of the response to it receiving an ACK for the last byte.

The table below shows the percentiles and average latency improvement of key Web applications, including even those responses without losses, measured over a period of one week. The key takeaway is: the average response time improved up to 7% and the 99th percentile improved by 10%. Nearly all of the improvement for TLP is in the tail latency (post-90th percentile). The varied improvements across services are due to different response-size distributions and traffic patterns. For example, TLP helps the most for Images, as these are served by multiple concurrently active TCP connections which increase the chances of tail segment losses.

Application	Average	99%
Google Web Search	-3%	-5%
Google Maps	-5%	-10%
Google Images	-7%	-10%

TLP also improved performance in mobile networks -- by 7.2% for Web search and Instant and 7.6% for Images transferred over Verizon network. To see why and where the latency improvements are coming from, we measured the retransmission statistics. We broke down the retransmission stats based on nature of retransmission -- timeout retransmission or fast recovery. TLP reduced the number of timeouts by 15% compared to the baseline, i.e.  $(\text{timeouts\_tlp} - \text{timeouts\_baseline}) / \text{timeouts\_baseline} = 15\%$ . Instead, these losses were either recovered via fast recovery or by the loss probe retransmission itself. The largest reduction in timeouts is when the sender is in the Open state in which it receives only insequence ACKs and no duplicate ACKs, likely because of tail losses. Correspondingly, the retransmissions occurring in the slow start phase after RTO reduced by 46% relative to baseline. Note that it is not always possible for TLP to convert 100% of the timeouts into fast recovery episodes because a probe itself may be lost. Also notable in our experiments is a significant decrease in the number of spurious timeouts -- the experiment had 61% fewer congestion window undo events. The Linux TCP sender uses either DSACK or timestamps to determine if retransmissions are spurious and employs techniques for undoing congestion window reductions. We also note that the total number of retransmissions decreased 7% with TLP because of the decrease in spurious retransmissions, and because the TLP probe itself plugs a hole.

We also quantified the overhead of probe packets. The probes accounted for 0.48% of all outgoing segments, i.e.  $(\text{number of probe segments} / \text{number of outgoing segments}) * 100 = 0.48\%$ . This is a reasonable overhead when contrasted with the overall retransmission rate of 3.2%. 10% of the probes sent are new segments and the rest are retransmissions, which is unsurprising given that short Web responses often don't have new data to send. We also found that in about 33% of the cases, the probes themselves plugged the only hole at receiver and the loss detection algorithm reduced the congestion window. 37% of the probes were not necessary and resulted in a duplicate acknowledgment.

Besides the macro level latency and retransmission statistics, we report some measurements from TCP's internal state variables at the

point when a probe segment is transmitted. The following distribution shows the FlightSize and congestion window values when a PTO is scheduled. We note that cwnd is not the limiting factor and that nearly all of the probe segments are sent within the congestion window.

percentile	10%	25%	50%	75%	90%	99%
FlightSize	1	1	2	3	10	20
cwnd	5	10	10	10	17	44

We have also experimented with a few variations of TLP: multiple probe segments versus single probe for the same tail loss episode, and several values for WCDelAckT. Our experiments show that sending just one probe suffices to get most (~90%) of latency benefits. The experiment results reported in this section and our current implementation limits number of probes to one, although the draft itself allows up to two consecutive probes. We chose the worst case delayed ack timer to be 200ms. When FlightSize equals 1 it is important to account for the delayed ACK timer in the PTO value, in order to bring down the number of unnecessary probe segments. With delays of 0ms and 50ms, the probe overhead jumped from 0.48% to 3.1% and 2.2% respectively. We have also experimented with transmitting 1-byte probe retransmissions as opposed to an entire MSS retransmission probe. While this scheme has the advantage of not requiring the loss detection algorithm outlined in Section 3, it turned out to be problematic to implement correctly in certain TCP stacks. Additionally, retransmitting 1-byte probe costs one more RTT to recover single packet tail losses, which is detrimental for short transfer latency.

## 6. Related work

TCP's long and conservative RTO recovery has long been identified as the major performance bottleneck for latency-demanding applications. A well-studied example is online gaming that requires reliability and low latency but small bandwidth. [GRIWODZ06] shows that repeated long RTO is the dominating performance bottleneck for game responsiveness. The authors in [PETLUND08] propose to use linear RTO to improve the performance, which has been incorporated in the Linux kernel as a non-default socket option for such thin streams. [MONDAL08] further argues exponential RTO backoff should be removed because it is not necessary for the stability of Internet. In contrast, TLP does not change the RTO timer calculation or the exponential back off. TLP's approach is to keep the behavior after RTO conservative for stability but allows a few timely probes before concluding the network is badly congested and cwnd should fall to 1.

As noted earlier in the Introduction the F-RTO [RFC5682] algorithm reduces the number of spurious timeout retransmissions and the Early Retransmit [RFC5827] mechanism reduces timeouts when a connection has received a certain number of duplicate ACKs. Both are complementary to TLP and can work alongside. Rescue retransmission introduced in [RFC6675] deals with loss events such as AL\*SL\* (using the same notation as section 4). TLP covers wider range of events such as AL\*. We experimented with rescue retransmission on Google Web servers, but did not observe much performance improvement. When the last segment is lost, it is more likely that a number of contiguous segments preceding the segment are also lost, i.e. AL\* is common. Timeouts that occur in the fast recovery are rare.

[HURTIG13] proposes to offset the elapsed time of the pending packet when re-arming the RTO timer. It is possible to apply the same idea for the TLP timer as well. We have not yet tested such a change to TLP.

Tail Loss Probe is one of several algorithms designed to maximize the robustness of TCPs self clock in the presence of losses. It follows the same principles as Proportional Rate Reduction [IMC11PRR] and TCP Laminar [Laminar].

On a final note we note that Tail loss probe does not eliminate 100% of all RTOs. RTOs still remain the dominant mode of loss recovery for short transfers. More work in future should be done along the following lines: transmitting multiple loss probes prior to finally resorting to RTOs, maintaining ACK clocking for short transfers in the absence of new data by clocking out old data in response to incoming ACKs, taking cues from applications to indicate end of transactions and use it for smarter tail loss recovery.

## 7. Security Considerations

The security considerations outlined in [RFC5681] apply to this document. At this time we did not find any additional security problems with Tail loss probe.

## 8. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

## 9. References

- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 2012.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.
- [RFC5827] Allman, M., Ayesta, U., Wang, L., Blanton, J., and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)", RFC 5827, April 2010.
- [RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682, September 2009.
- [IMC11PRR] Mathis, M., Dukkkipati, N., Cheng, Y., and M. Ghobadi, "Proportional Rate Reduction for TCP", Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference , 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [FACK] Mathis, M. and M. Jamshid, "Forward acknowledgement: refining TCP congestion control", ACM SIGCOMM Computer Communication Review, Volume 26, Issue 4, Oct. 1996. , 1996.
- [RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.
- [RFC2018] Mathis, M. and J. Mahdavi, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [GRIWODZ06] Griwodz, C. and P. Halvorsen, "The fun of using TCP for an MMORPG", NOSSDAV , 2006.

- [PETLUND08] Petlund, A., Evensen, K., Griwodz, C., and P. Halvorsen, "TCP enhancements for interactive thin-stream applications", NOSSDAV , 2008.
- [MONDAL08] Mondal, A. and A. Kuzmanovic, "Removing Exponential Backoff from TCP", ACM SIGCOMM Computer Communication Review , 2008.
- [Laminar] Mathis, M., "Laminar TCP and the case for refactoring TCP congestion control", July 2012.
- [HURTIG13] Hurtig, P., Brunstrom, A., Petlund, A., and M. Welzl, "TCP and SCTP RTO Restart", draft-ietf-tcpm-rtorestart-00 (work in progress), February 2013.

## Authors' Addresses

Nandita Dukkipati  
Google, Inc  
1600 Amphitheater Parkway  
Mountain View, California 93117  
USA

Email: nanditad@google.com

Neal Cardwell  
Google, Inc  
76 Ninth Avenue  
New York, NY 10011  
USA

Email: ncardwell@google.com

Yuchung Cheng  
Google, Inc  
1600 Amphitheater Parkway  
Mountain View, California 93117  
USA

Email: ycheng@google.com

Matt Mathis  
Google, Inc  
1600 Amphitheater Parkway  
Mountain View, California 93117  
USA

Email: mattmathis@google.com



TCPM working group  
Internet Draft  
Intended Status: Informational  
Expires: 11/7/2015

M. Fox  
C. Kassimis  
J. Stevens  
IBM  
May 7, 2015

IBM's Shared Memory Communications over RDMA  
draft-fox-tcpm-shared-memory-rdma-07.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on November 7, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Abstract

This document describes the IBM's Shared Memory Communications over RDMA (SMC-R) protocol. This protocol provides RDMA communications to TCP endpoints in a manner that is transparent to socket applications. It further provides for dynamic discovery of partner RDMA capabilities and dynamic setup of RDMA connections, transparent high availability and load balancing when redundant RDMA network paths are available, and it maintains many of the traditional TCP/IP qualities of service such as filtering that enterprise users demand, as well as TCP socket semantics such as urgent data.

## Table of Contents

1. Introduction.....	5
1.1. Summary of changes in this draft.....	6
1.2. Protocol overview.....	6
1.2.1. Hardware requirements.....	8
1.3. Definition of common terms.....	8
2. Link Architecture.....	10
2.1. Remote Memory Buffers (RMBs).....	12
2.2. SMC-R Link groups.....	16
2.2.1. Link group types.....	17
2.2.2. Maximum number of links in link group.....	20
2.2.3. Forming and managing link groups.....	21
2.2.4. SMC-R link identifiers.....	22
2.3. SMC-R resilience and load balancing.....	23
3. SMC-R Rendezvous architecture.....	25
3.1. TCP options.....	25
3.2. Connection Layer Control (CLC) messages.....	26
3.3. LLC messages.....	26
3.4. CDC Messages.....	28
3.5. Rendezvous flows.....	28
3.5.1. First contact.....	28
3.5.1.1. TCP Options pre-negotiation.....	28
3.5.1.2. Client Proposal.....	29
3.5.1.3. Server acceptance.....	30
3.5.1.4. Client confirmation.....	31
3.5.1.5. Link (QP) confirmation.....	31
3.5.1.6. Second SMC-R link setup.....	34
3.5.1.6.1. Client processing of "Add Link" LLC message from server.....	34
3.5.1.6.2. Server processing of "Add Link" reply LLC message from the client.....	35

3.5.1.6.3.	Exchange of Rkeys on second SMC-R link....	37
3.5.1.6.4.	Aborting SMC-R and falling back to IP.....	37
3.5.2.	Subsequent contact.....	37
3.5.2.1.	SMC-R proposal.....	38
3.5.2.2.	SMC-R acceptance.....	39
3.5.2.3.	SMC-R confirmation.....	40
3.5.2.4.	TCP data flow race with SMC Confirm CLC message	40
3.5.3.	First contact variation: creating a parallel link group	41
3.5.4.	Normal SMC-R link termination.....	42
3.5.5.	Link group management flows.....	43
3.5.5.1.	Adding and deleting links in an SMC-R link group	43
3.5.5.1.1.	Server initiated Add Link processing.....	43
3.5.5.1.2.	Client initiated Add Link processing.....	44
3.5.5.1.3.	Server initiated Delete Link Processing...	44
3.5.5.1.4.	Client initiated Delete Link request.....	46
3.5.5.2.	Managing multiple Rkeys over multiple SMC-R links	48
in a link group.....		48
3.5.5.2.1.	Adding a new RMB to an SMC-R link group...	49
3.5.5.2.2.	Deleting an RMB from an SMC-R link group..	52
3.5.5.2.3.	Adding a new SMC-R link to a link group with	53
multiple RMBs.....		53
3.5.5.3.	Serialization of LLC exchanges, and collisions.	54
3.5.5.3.1.	Collisions with ADD LINK / CONFIRM LINK	56
exchange.....		56
3.5.5.3.2.	Collisions during DELETE LINK exchange....	57
3.5.5.3.3.	Collisions during CONFIRM_RKEY exchange...	57
4.	SMC-R memory sharing architecture.....	59
4.1.	RMB element allocation considerations.....	59
4.2.	RMB and RMBE format.....	59
4.3.	RMBE control information.....	59
4.4.	Use of RMBEs.....	60
4.4.1.	Initializing and accessing RMBEs.....	60
4.4.2.	RMB element reuse and conflict resolution.....	61
4.5.	SMC-R protocol considerations.....	62
4.5.1.	SMC-R protocol optimized window size updates.....	62
4.5.2.	Small data sends.....	63
4.5.3.	TCP Keepalive processing.....	63
4.6.	TCP connection failover between SMC-R links.....	66
4.6.1.	Validating data integrity.....	66
4.6.2.	Resuming the TCP connection on a new SMCR link.....	67
4.7.	RMB data flows.....	67
4.7.1.	Scenario 1: Send flow, window size unconstrained....	68
4.7.2.	Scenario 2: Send/Receive flow, window unconstrained.	70
4.7.3.	Scenario 3: Send Flow, window constrained.....	71
4.7.4.	Scenario 4: Large send, flow control, full window size	73
writes.....		73

4.7.5. Scenario 5: Send flow, urgent data, window size unconstrained.....	76
4.7.6. Scenario 6: Send flow, urgent data, window size closed	78
4.8. Connection termination.....	80
4.8.1. Normal SMC-R connection termination flows.....	80
4.8.1.1. Abnormal SMC-R connection termination flows....	85
4.8.1.2. Other SMC-R connection termination conditions..	87
5. Security considerations.....	88
5.1. VLAN considerations.....	88
5.2. Firewall considerations.....	88
5.3. Host-based IP Filters.....	89
5.4. Intrusion Detection Services.....	89
5.5. IP Security (IPSec).....	89
5.6. TLS/SSL.....	89
6. IANA considerations.....	89
7. References.....	90
7.1. Normative References.....	90
7.2. Informative References.....	90
8. Acknowledgments.....	90
9. Conventions used in this document.....	90
Appendix A. Formats.....	91
A.1. TCP option.....	91
A.2. CLC messages.....	91
A.2.1. Peer ID format.....	91
A.2.2. SMC Proposal CLC message format.....	93
A.2.3. SMC Accept CLC message format.....	96
A.2.4. SMC Confirm CLC message format.....	99
A.2.5. SMC Decline CLC message format.....	102
A.3. LLC messages.....	103
A.3.1. CONFIRM LINK LLC message format.....	104
A.3.2. ADD LINK LLC message format.....	106
A.3.3. ADD LINK CONTINUATION LLC message format.....	108
A.3.4. DELETE LINK LLC message format.....	111
A.3.5. CONFIRM RKEY LLC message format.....	113
A.3.6. CONFIRM RKEY CONTINUATION LLC message format.....	116
A.3.7. DELETE RKEY LLC message format.....	118
A.3.8. TEST LINK LLC message format.....	120
Appendix B. Socket API considerations.....	126
Appendix C. Rendezvous Error scenarios.....	128
C.1. SMC Decline during CLC negotiation.....	128
C.2. SMC Decline during LLC negotiation.....	128
C.3. The SMC Decline window.....	130
C.4. Out of synch conditions during SMC-R negotiation.....	130
C.5. Timeouts during CLC negotiation.....	131
C.6. Protocol errors during CLC negotiation.....	131
C.7. Timeouts during LLC negotiation.....	132
C.7.1. Recovery actions for LLC timeouts and failures.....	133

C.8. Failure to add second SMC-R link to a link group.....140

1. Introduction

This document specifies IBM's Shared Memory Communications over RDMA (SMC-R) protocol. SMC-R is a protocol for Remote Direct Memory Access (RDMA) communication between TCP socket endpoints. SMC-R runs over networks that support RDMA over Converged Ethernet (RoCE). It is designed to permit existing TCP applications to benefit from RDMA without requiring modifications to the applications or predefinition of RDMA partners.

SMC-R provides dynamic discovery of the RDMA capabilities of TCP peers and automatic setup of RDMA connections that those peers can use. SMC-R also provides transparent high availability and load balancing capabilities that are demanded by enterprise installations but are missing from current RDMA protocols. If redundant RoCE capable hardware such as RDMA NICs (RNICs) and RoCE capable switches is present, SMC-R can load balance over that redundant hardware and can also non-disruptively move TCP traffic from failed paths to surviving paths, all seamlessly to the application and the sockets layer. Because SMC-R preserves socket semantics and the TCP three-way handshake, many TCP qualities of service such as filtering, load balancing, and SSL encryption are preserved, as are TCP features such as urgent data.

Because of the dynamic discovery and setup of SMC-R connectivity between peers, no RDMA connection manager (RDMA-CM) is required. This also means that support for UD queue pairs is also not required.

It is recommended that the SMC-R services be implemented in kernel space, which enables optimizations such as resource sharing between connections across multiple processes and also permits applications using SMC-R to spawn multiple processes (e.g. fork) without losing SMC-R functionality. A user space implementation is compatible with this architecture, but it may not support spawned processes (i.e. fork) which limits sharing and resource optimization to TCP connections that originate from the same process. This might be an appropriate design choice if the use case is a system that hosts a large single process application that creates many TCP connections to a peer host, or in implementations where a kernel space implementation is not possible or introduces excessive overhead for kernel space to user space context switches.

### 1.1. Summary of changes in this draft

Changed the title to add "IBM's".

### 1.2. Protocol overview

SMC-R defines the concept of the SMC-R Link, which is a logical point-to-point link using reliably connected queue pairs between TCP/IP stack peers over a RoCE fabric. An SMC-R link is bound to a specific hardware path, meaning a specific RNIC on each peer. SMC-R links are created and maintained by an SMC-R layer, which may reside in kernel or user space depending upon operating system and implementation requirements. The SMC-R layer resides below the sockets layer and directs data traffic for TCP connections between connected peers over the RoCE fabric using RDMA rather than over a TCP connection. The TCP/IP stack with its fragmentation, packetization, etc. requirements is bypassed and the application data is moved between peers using RDMA.

Multiple SMC-R links between the same two TCP/IP stack peers are also supported. A set of SMC-R links called a link group can be logically bonded together to provide redundant connectivity. If there is redundant hardware, for example two RNICs on each peer, separate SMC-R links are created between the peers to exploit that redundant hardware. The link group architecture with redundant links provide load balancing, increased bandwidth as well as seamless failover.

Each SMC-R link group is associated with an area of memory called Remote Memory Buffers (RMBs), which are areas of memory that are available for SMC-R peers to write into using RDMA writes. Multiple TCP connections between peers may be multiplexed over a single SMC-R link, in which case the SMC-R layer manages the partitioning of the RMBs between the TCP connections. This multiplexing reduces the RDMA resources such as queue pairs and RMBs that are required to support multiple connections between peers, and also reduces the processing and delays related to setting up queue pairs, pinning memory, and other RDMA setup tasks when new TCP connections are created. In a kernel space SMC-R implementation in which the RMBs reside in kernel storage, this sharing and optimization works across multiple processes executing on the same host. In a user space SMC-R implementation in which the RMBs reside in user space, this sharing

and optimization is limited to multiple TCP connections created by a single process, as separate RMBs and QPs will be required for each process.

SMC-R also introduces a rendezvous protocol that is used to dynamically discover the RDMA capabilities of TCP connection partners and exchange credentials necessary to exploit that capability if present. TCP connections are set up using the normal TCP 3-way handshake, with the addition of a new TCP option that indicates SMC-R capability. If both partners indicate SMC-R capability then at the completion of the 3-way TCP handshake the SMC-R layers in each peer take control of the TCP connection and use it to exchange additional connection level control (CLC) messages to negotiate SMC-R credentials such as queue pair (QP) information, addressability over the RoCE fabric, RMB buffer sizes, keys and addresses for accessing RMBs over RDMA, etc. If at any time during this negotiation a failure or decline occurs, the TCP connection falls back to using the IP fabric.

If the SMC-R negotiation succeeds and either a new SMC-R link is set up or an existing SMC-R link is chosen for the TCP connection, then the SMC-R layers open the sockets to the applications and the applications use the sockets as normal. The SMC-R layer intercepts the socket reads and writes and moves the TCP connection data over the SMC-R link, "out of band" to the TCP connection which remains open and idle over the IP fabric, except for termination flows and possible keepalive flows. Regular TCP sequence numbering methods are used for the TCP flows that do occur; data flowing over RDMA does not use or affect TCP sequence numbers.

This architecture does not support fallback of active SMC-R connections to IP. Once connection data has completed the switch to RDMA, a TCP connection cannot be switched back to IP and will reset if RDMA becomes unusable.

The SMC-R protocol defines the format of the Remote Memory Buffers that are used to receive TCP connection data written over RDMA, as well as the semantics for managing and writing to these buffers using Connection Data Control (CDC) messages.

Finally, SMC-R defines link level control (LLC) messages that are exchanged over the RoCE fabric between peer SMC-R layers to manage the SMC-R links and link groups. These include messages to test and confirm connectivity over an SMC-R link, add and delete SMC-R links to or from the link group, and exchange RMB addressability information.

### 1.2.1. Hardware requirements

SMC-R does not require full Converged Enhanced Ethernet switch functionality. SMC-R functions over standard Ethernet fabrics provided endpoint RNICs are provided and IEEE 802.3x Global Pause Frame is supported and enabled in the switch fabric.

While SMC-R as specified in this document is designed to operate over RoCE fabrics, adjustments to the rendezvous methods could enable it to run over other RDMA fabrics such as Infiniband and iWARP.

### 1.3. Definition of common terms

This section provides definitions of terms that have a specific meaning to the SMC-R protocol and are used throughout this document.

#### SMC-R link

An SMC-R Link is a logical point to point connection over the RoCE fabric via specific physical adapters (MAC/GID). The Link is formed during the first contact sequence of the TCP/IP 3 way handshake sequence that occurs over the IP fabric. During this handshake an RDMA RC-QP connection is formed between the two peer SMC hosts and is defined as the SMC Link. The SMC Link can then support multiple TCP connections between the two peers. An SMC link is associated with a single LAN (or VLAN) segment and is not routable.

#### SMC-R link group

An SMC-R Link Group is a group of SMC-R Links typically each over unique RoCE adapters between the same two SMC-R peers. Each link in the link group has equal characteristics such as the same VLAN ID (if VLANs are in use), access to the same RMB(s) and the same TCP server / client

#### SMC-R peer

The SMC-R Peer is the peer software stack within the peer Operating System with respect the Shared Memory Communications (messaging) protocol.

#### SMC-R Rendezvous

The SMC-R Rendezvous is the SMC-R peer discovery and handshake sequence that occurs transparently over the IP (Ethernet) fabric during and immediately after the TCP connection 3 way handshake

by exchanging the SMC capabilities and credentials using experimental TCP option and CLC messages.

#### TCP Client

The TCP socket-based peer that initiates a TCP connection

#### TCP Server

The TCP socket-based peer that accepts a TCP connection

#### CLC messages

The SMC-R protocol defines a set of Connection Layer Control Messages that flow over the TCP connection that are used to manage SMC link rendezvous at TCP connection setup time. This mechanism is analogous to SSL setup messages

#### LLC Commands

The SMC-R protocol defines a set of RoCE Link Layer Control Commands that flow over the RoCE fabric using RDMA sendmsg, that are used to manage SMC Links, SMC Link Groups and SMC Link Group RMB expansion and contraction.

#### CDC message

The SMC-R protocol defines a Connection Data Control message that flows over the RoCE fabric using RDMA sendmsg that is used to manage the SMC-R connection data. This message provides information about data being transferred over the out of band RDMA connection, such as data cursors, sequence numbers, and data flags (for example urgent data). The receipt of this message also provides an interrupt to inform the receiver that it has received RDMA data.

#### RMB

A Remote (RDMA) Memory Buffer is a fixed or pinned buffer allocated in each of the peer hosts for a TCP (via SMC-R) connection. The RMB is registered to the RNIC and allows remote access by the remote peer using RDMA semantics. Each host is passed the peer's RMB specific access information (RKey and RMB Element offset) during the SMC-R rendezvous process. The host stores socket application user data directly into the peer's RMB using RDMA over RoCE.

#### Rtoken

The combination of an RMB's Rkey and RDMA virtual addressing, an Rtoken provides addressability to an RMB to an RDMA peer

#### RMBE

The Remote Memory Buffer Element is an area of an RMB that is allocated to a specific TCP connection. The RMBE contains data for the TCP connection. The RMBE represents the TCP receive buffer whereby the remote peer writes into the RMBE and the local peer reads from the local RMBE. The alert token resolves to a specific RMBE.

#### Alert Token

The SMC-R alert token is a four byte value that uniquely identifies the TCP connection over an SMC-R connection. The alert token allows the SMC peer to quickly identify the target TCP connection that now has new work. The format of the token is defined by the owning SMC-R end point and is considered opaque to the remote peer. However the token should not simply be an index to an RMBE element; it should reference a TCP connection and be able to be validated to avoid reading data from stale connections.

#### RNIC

The RDMA capable Network Interface Card (RNIC) is an Ethernet NIC that supports RDMA semantics and verbs using RoCE.

#### First Contact

Describes an SMC-R negotiation to set up the first link in a link group

#### Subsequent Contact

Describes an SMC-R negotiation between peers who are using an already existing SMC-R link group

## 2. Link Architecture

An SMC-R link is based on reliably connected queue pairs (QPs) that form a "logical point to point link" between the two SMC-R peers over a RoCE fabric. An SMC-R link extends from SMC-R peer to SMC-R peer,

where typically each peer would be a TCP/IP stack would reside on separate hosts.

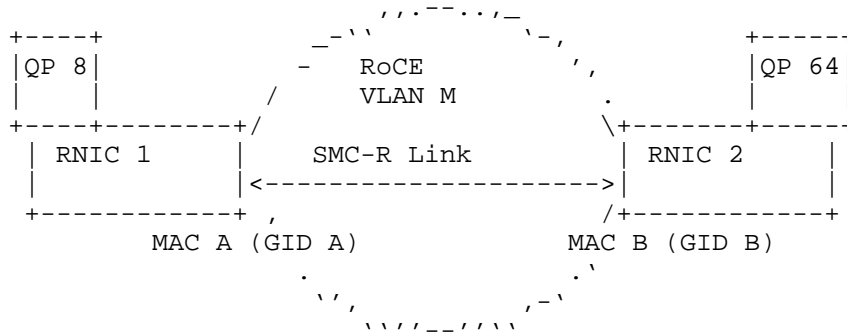


Figure 1 SMC-R Link Overview

Figure 1 illustrates an overview of the basic concepts of SMC-R peer to peer connectivity which is called the SMC-R Link. The SMC-R Link forms a logical point to point connection between two SMC-R peers via RoCE. The SMC Link is defined and identified by the following attributes:

SMC-R Link = RC QPs (source VMAC GID QP + target VMAC GID QP + VLAN ID)

The SMC-R Link can optionally be associated with a VLAN ID. If VLANs are in use for the associated IP (LAN) connection then the VLAN attribute is carried over on the SMC-R link. When VLANs are in use each SMC-R link group is associated with a single and specific VLAN. The RoCE fabric is the same physical Ethernet LAN used for standard TCP/IP over Ethernet communications, with switches as described in 1.2.1.

An SMC-R Link is designed to support multiple TCP connections between the same two peers. An SMC Link is intended to be long lived while the underlying TCP connections can dynamically come and go. The associated RMBs can also be dynamically added and removed from the link as needed. The first TCP connection between the peers establishes the SMC-R link. Subsequent TCP connections then use the previously established link. When the last TCP connection terminates the link can then be terminated, typically after an implementation defined idle time-out period has elapsed. The TCP server is responsible for initiating and terminating the SMC Link.

## 2.1. Remote Memory Buffers (RMBs)

Figure 2 shows the hosts X and Y and their associated RMBs within each host. With the SMC-R link and the associated RMB keys (Rkeys) and RDMA virtual addresses each SMC-R enabled TCP/IP stack can remotely access its peer's RMBs using RDMA. The RKeys and virtual addresses are exchanged during the rendezvous processing when the link is established. The combination of the Rkey and the virtual address is the Rtoken. Note that the SMC-R Link ends at the QP providing access to the RMB (via the Link + RToken).

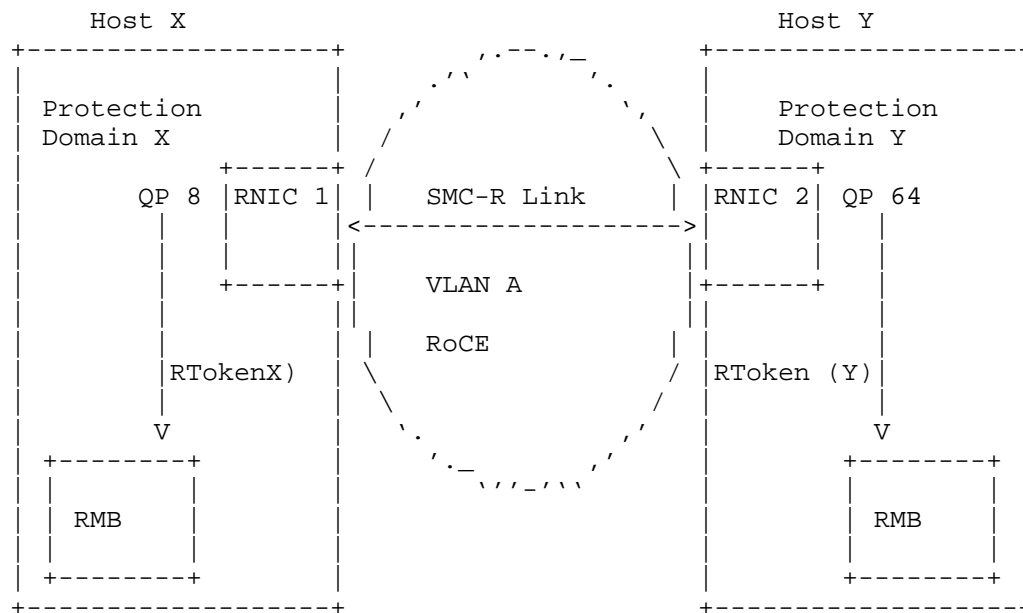


Figure 2 SMC link and RMBs

An SMC-R link can support multiple RMBs which are independently managed by each peer. The number of and the size of RMBs are managed by the peers based on host unique memory management requirements; however the maximum number of RMBs that can be associated to a link group on one peer is 255. The QP has a single protection domain, but each RMB has a unique RToken. All RTokens must be exchanged with the peer.

Each peer manages the RMBs in its local memory for its remote SMC-R peer by sharing access to the RMBs via Rtokens with its peers. The remote peer writes into the RMBs via RDMA and the local peer (RMB owner) then reads from the RMBs.

When two peers decide to use SMC-R for a given TCP connection, they each allocate a local RMB Element for the TCP connection and communicate the location of this local RMB Element during rendezvous processing. To that end, RMB elements are created in pairs, with one RMB element allocated locally on each peer of the SMC-R link.

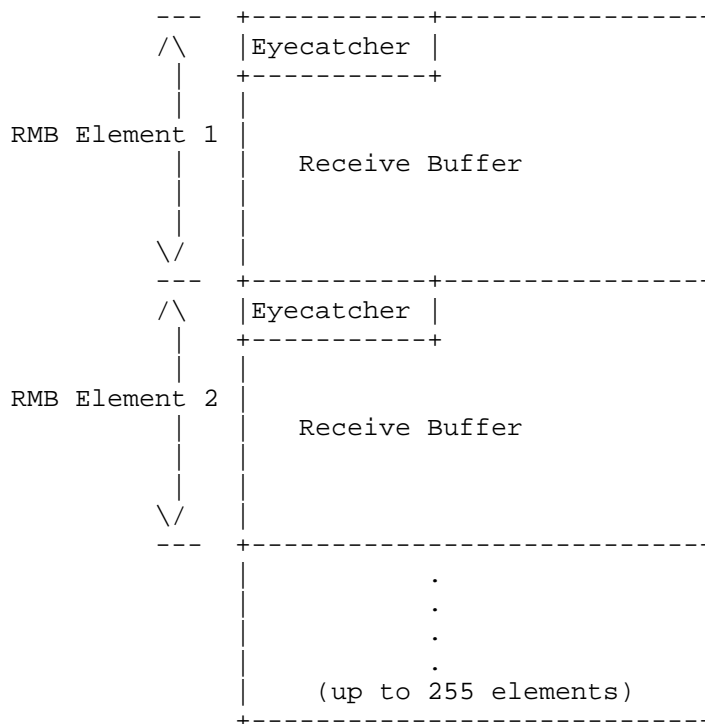


Figure 3 RMB Format

Figure 3 illustrates the basic format of an RMB. The RMB is a virtual memory buffer whose backing real memory is pinned, which can support up to 255 TCP connections to exactly one remote SMC-R peer. Each RMB is therefore associated with the SMC-R links within a link group for the two peers and a specific RoCE Protection Domain. Other than the 2 peers identified by the SMC-R link no other SMC-R peers can have RDMA access to an RMB; this requires a unique Protection Domain for every SMC-R Link. This is critical to ensure integrity of SMC-R communications.

RMBs are subdivided into multiple elements for efficiency, with each RMBE element (RMBE) is associated with a single TCP connection. Therefore multiple TCP connections across an SMC link group can share the same memory for RDMA purposes, reducing the overhead of having to register additional memory with the RNIC for every new TCP connection. The number of elements in an RMB and the size of each RMB Element is entirely governed by the owning peer subject to the SMC-R architecture rules, however, all RMB elements within a given RMB must be the same size. Each peer can decide the level of resource sharing that is desirable across TCP connections based on local constraints such as available system memory, etc. An RMB Element is identified to the remote SMC-R peer via an RMB Element Token which consists of the following:

- o RMB RToken: The combination of the Rkey and virtual address provided by the RNIC that identifies the start of the RMB for RDMA operations.
- o RMB Index: Identifies the RMB element index in the RMB. Used to locate a specific RMB element within an RMB. Valid value range is 1-255.
- o RMB element length: The length of the RMB element's eyecatcher plus the length of receive buffer. This length is equal for all RMB elements in a given RMB. This length can be variable across different RMBs.

Multiple RMBs can be associated to an SMC-R link group and each peer in an SMC-R link group manages allocation of its RMBs. RMB allocation can be asymmetric. For example, server X can allocate 2 RMBs to an SMC-R link group while server Y allocates 5. This provides maximum implementation flexibility to allow hosts optimize RMB management for their own local requirements. The maximum number of RMBs that can be allocated on one peer to a link group is 255. If more RMBs are required, the peer may fall back to IP for subsequent connections or, if the peer is the server, create a parallel link group.

One use case for multiple RMBs is multiple receive buffer sizes. Since every element in an RMB must be the same size, multiple RMBs with different element sizes can be allocated if varying receive buffer sizes are required.

Also since the maximum number of TCP connections whose receive buffers can be allocated to an RMB is 255, multiple RMBs may be required to provide capacity for large numbers of TCP connections between two peers.

Separately from the RMB, the TCP/IP stack that owns each RMB maintains control data for each RMB element within its local control structures. The control data contains flags for maintaining the state of the TCP data (for example, urgent indicator) and most importantly, two cursors which are illustrated in Figure 4:

- o The peer producer cursor: This is a wrapping offset into the RMB element's receive buffer that points to the next byte of data to be written by the remote peer. This cursor is provided by the remote peer in a Connection Data Control (CDC message), which is sent using RDMA sendmsg processing, and tells the local peer how far it can consume data in the RMBE buffer.
- o The peer consumer cursor: This is a wrapping offset into the remote peer's RMB element's receive buffer that points to the next byte of data to be consumed by the remote peer in its own RMBE. The local cannot write into the remote peer's RMBE beyond this point without causing data loss. This cursor is also provided by the peer using a Connection Data Control message.

Each TCP connection peer maintains its cursors for a TCP connection's RMBE in its local control structures. In other words, the peer who writes into a remote peer's RMBE provides its producer cursor to the peer whose RMBE it has written into. The peer who reads from its RMBE provides its consumer cursor to the writing peer. In this manner the reads and writes between peers are kept coordinated.

For example, referring to Figure 4, peer B writes the hashed data into the receive buffer of peer A's RMBE. After that write completes, peer B uses a CDC message to update its producer cursor to peer A, to indicate to peer A how much data is available for peer A to consume. The CDC message that peer B sends to peer A wakes up peer A and notifies it that there is data to be consumed.

Similarly, when peer A consumes data written by peer B, it uses a CDC message to update its consumer cursor to peer B to let peer B know how much data it has consumed, so peer B knows how much space is available for further writes. If peer B were to write enough data to peer A that it would wrap the RMBE receive buffer and exceed the consumer cursor, data loss would result.

Note that this is a simplistic description of the control flows and they are optimized to minimize the number of CDC messages required, as described in 4.7. RMB data flows.

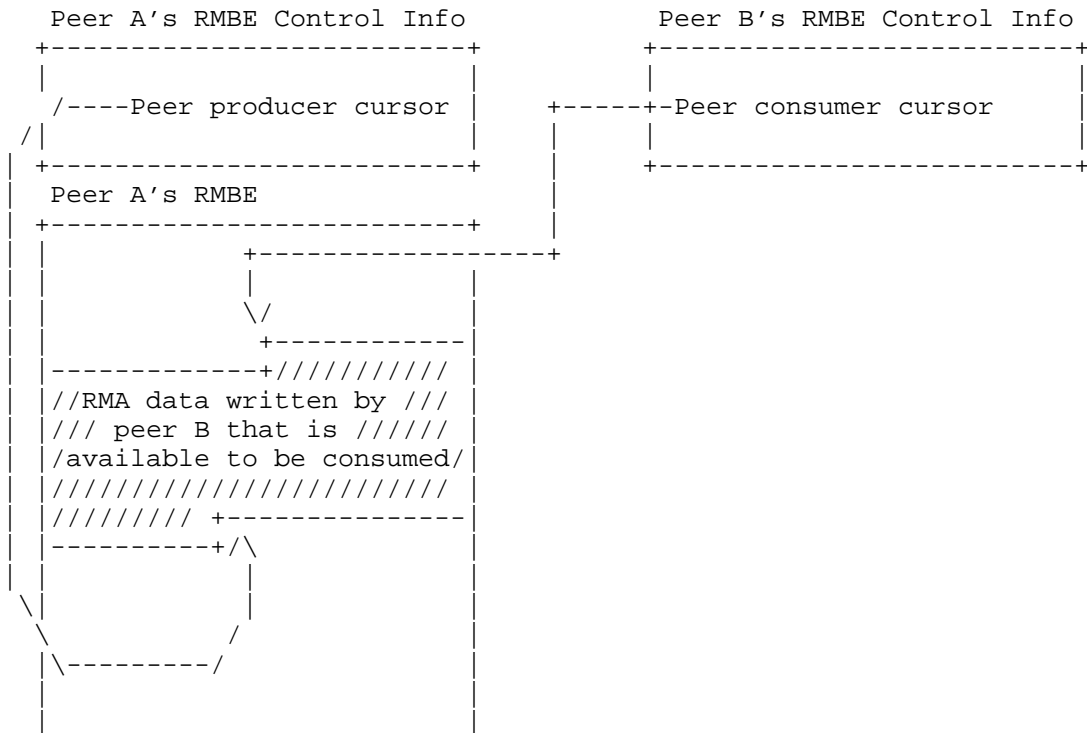


Figure 4 RMBE cursors

Additional flags and indicators are communicated between peers. In all cases, these flags and indicators are updated by the peer using CDC messages with the control information contained in inline data. More details on these additional flags and indicators are described in [4.3. RMBE control information](#).

## 2.2. SMC-R Link groups

SMC-R links are logically grouped together to form an SMC-R Link Group. The purpose of the Link Group is for supporting multiple links between the same two peers to provide for:

- o Resilience: Provides transparent and dynamic switching of the link used by existing TCP connections during link failures, typically hardware related. TCP traffic using the failing link can be switched to an active link within the link group avoiding disruptions to application workloads.

- o Link utilization: Provides an active/active link usage model allowing TCP traffic to be balanced across the links, which increases bandwidth and avoids hardware imbalances and bottlenecks. Note that both adapter and switch utilization can become potential resource constraint issues

SMC-R Link Group support is required. Resilience is not optional. However, the user can elect to provision a single RNIC (on one or both hosts).

Multiple links that are formed between the same two peers fall into two distinct categories:

1. Equal Links: Links providing equal access to the same RMB(s) at both endpoints whereby all TCP connections associated with the links must have the same VLAN ID and have the same TCP server and TCP client roles or relationship.
2. Unequal Links: Links providing access to unique, unrelated and isolated RMB(s) (i.e. for unique VLANs or unique and isolated application workloads, etc.) or have unique TCP server or client roles.

Links that are logically grouped together forming an SMC Link Group must be equal links.

#### 2.2.1. Link group types

Equal links within a link group also have another "Link Group Type" attribute based on the link's associated underlying physical path. The following SMC-R link types are defined:

1. Single Link: the only active link within a link group
2. Parallel Link: not allowed - SMC Links having the same physical RNIC at both hosts
3. Asymmetric Link: links that have unique RNIC adapters at one host but share a single adapter at the peer host
4. Symmetric Link: links that have unique RNIC adapters at both hosts

These link group types are further explained in the following figures and descriptions.

Figure 2 above shows the single link case. The single link illustrated in Figure 2 also establishes the SMC-R Link Group. Link groups are supposed to have multiple links, but when only one RNIC is available at both hosts then only a single link can be created. This is expected to be a transient case.

Figure 5 shows the symmetric link case. Both hosts have unique and redundant RNIC adapters. This configuration meets the objectives for providing full RoCE redundancy required to provide the level of resilience required for high availability for SMC-R. While this configuration is not required, it is a strongly recommended "best practice" for the exploitation of SMC-R. Single and asymmetric links must be supported but are intended to provide for short term transient conditions, for example during a temporary outage or recycle of a RNIC.

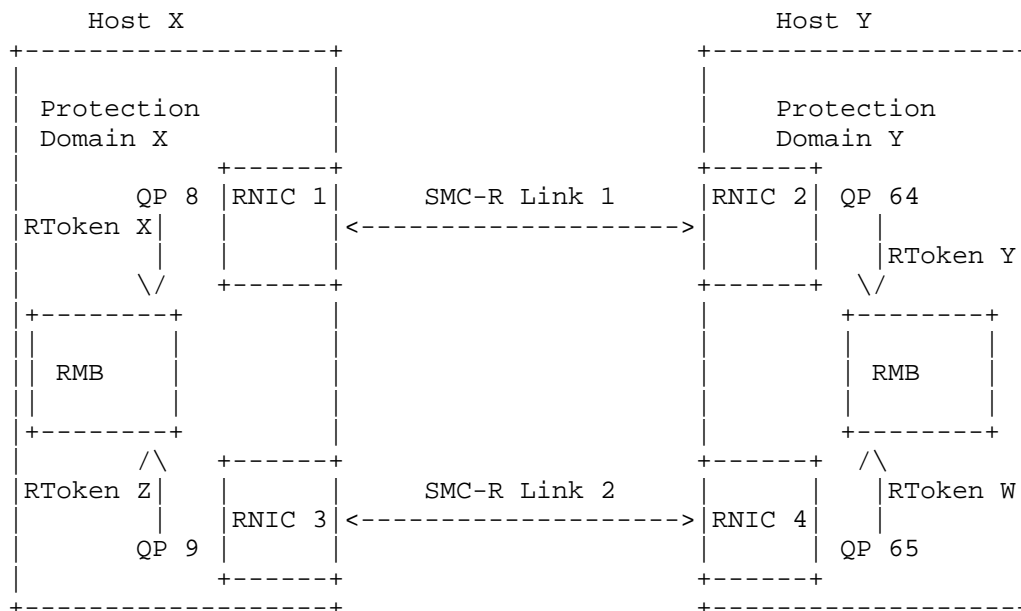


Figure 5 Symmetric SMC-R links

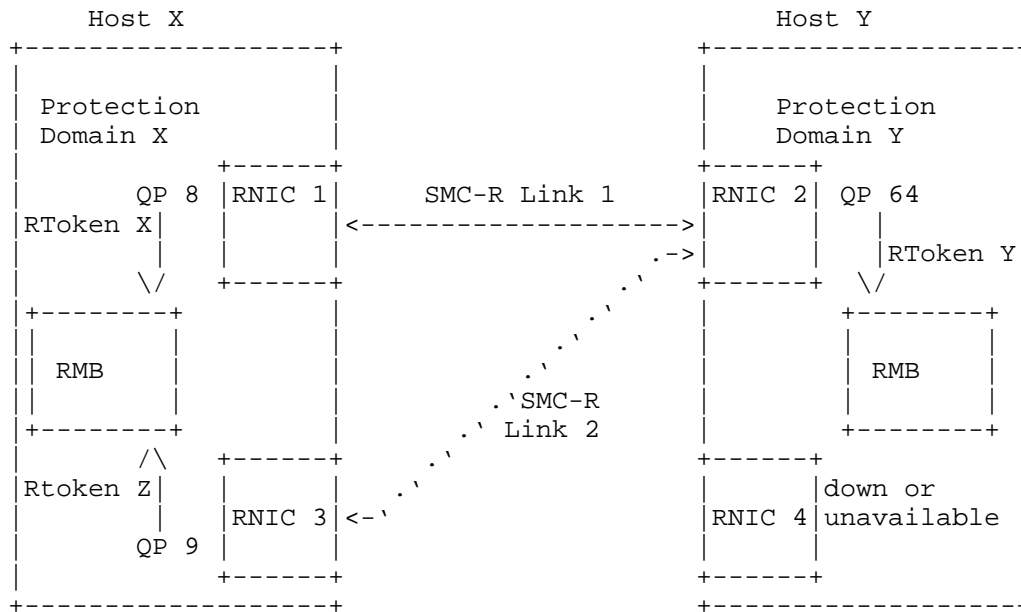


Figure 6 Asymmetric SMC-R links

In the example provided by Figure 6, host X has two RNICs but Host Y only has one RNIC. This configuration allows for the creation of an asymmetric link. While an asymmetric link will provide some resilience (i.e. when RNIC 1 fails) ideally each host should provide two redundant RNICs. This should be a transient case, and when RNIC 4 becomes available, this configuration must transition to a symmetric link configuration. This transition is accomplished by first creating the new symmetric link, then deleting the asymmetric link with reason code "Asymmetric link no longer needed" specified in the DELETE LINK LLC message.

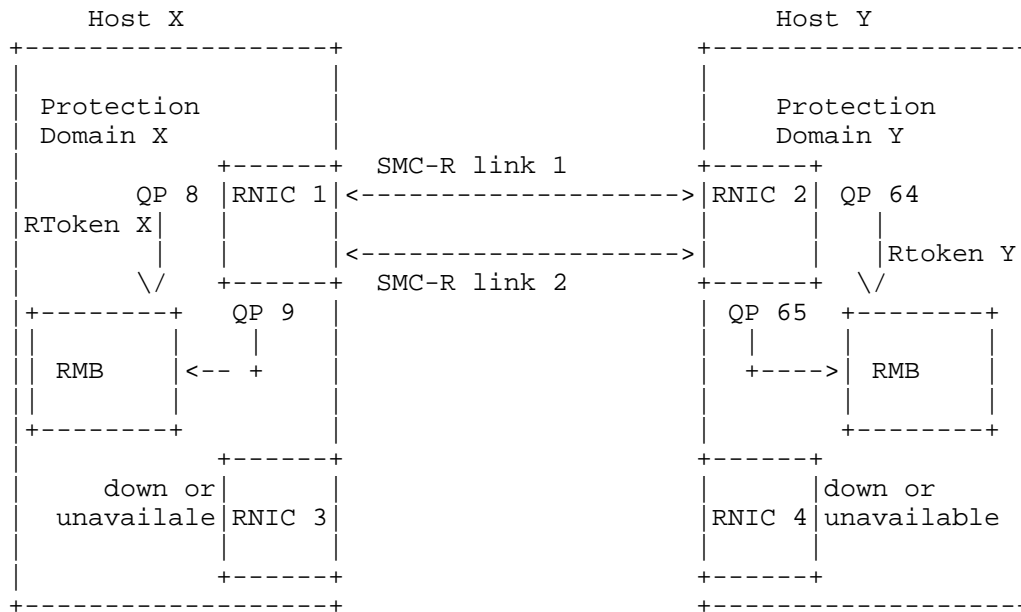


Figure 7 SMC-R parallel links (not supported)

Figure 7 shows parallel links, which are two links in the link group that use the same hardware. This configuration is not permitted. Because SMC-R multiplexes multiple TCP connections over an SMC-R link and both links are using the exact same hardware, there is no additional redundancy or capacity benefit obtained from this configuration. However this configuration does add unnecessary overhead of additional queue pairs, generation of additional Rkeys, etc.

#### 2.2.2. Maximum number of links in link group

The SMC-R protocol defines a maximum of 8 symmetric SMC-R links within a single SMC-R link group. This allows for support for up to 8 unique physical paths between peer hosts. However, in terms of meeting the basic requirements for redundancy support for at least 2 symmetric links must be implemented. Supporting greater than 2 links also simplifies implementation for practical matters relating to dynamically adding and removing links, for example starting a third SMC-R link prior to taking down one of the two existing links. Recall that all links within a link group must have equal access to all associated RMBs.

The SMC-R protocol allows an implementation to implement an implementation specific and appropriate value for maximum symmetric links. The implementation value must not exceed the architecture limit of 8 and the implementation must not be lower than 2, because the SMC-R protocol requires redundancy. This does not mean that two RNICs are physically required to enable SMC-R connectivity, but at least two RNICs for redundancy are strongly recommended.

The SMC-R peers exchange their implementation maximum link values during the link group establishment using the defined maximum link value in the CONFIRM LINK LLC command. Once the initial exchange completes the value is set for the life of the link group. The maximum link value can be provided by both the server and client. The server must supply a value, whereas the client maximum link value is optional. When the client does not supply a value, it indicates that the client accepts the server supplied maximum value. If the client provides a value it can not exceed the server maximum value. If the client passes a lower value then this lower value then becomes the final negotiated maximum number of symmetric links for this link group. Again, the minimum value is 2.

During run time the client must never request that the server add a symmetric link to a link group that would exceed the negotiated maximum link value. Likewise the server must never attempt to add a symmetric link to a link group that would exceed the negotiated maximum value.

In terms of counting the active link count within a link group, the initial link (or the only / last) link is always counted as 1. Then as additional links are added they are either symmetric or asymmetric links.

With regards to enforcing the maximum link rules, asymmetric links are an exception having a unique set of rules:

- o Asymmetric links are always limited to one asymmetric link allowed per link group
- o Asymmetric links must not be counted in the maximum symmetric link count calculation. When tracking the current count or enforcing the negotiated maximum number of links, an asymmetric link is not to be counted

### 2.2.3. Forming and managing link groups

SMC-R link groups are self-defining. The first SMC-R link in a link group is created using TCP option flows on the TCP three-way

handshake followed by CLC message flows over the TCP connection. Subsequent SMC-R links in the link group are created by sending LLC messages over an SMC-R link that already exists in the link group. Once an SMC-R link group is created, no additional SMC-R links in that group are created using TCP and CLC negotiation. Because subsequent SMC-R links are created exclusively by sending LLC messages over an existing SMC-R link in a link group, the membership of SMC-R links to a link group is self-defining.

This architecture does not define a specific identifier for an SMC-R link group. This identification may be useful for network management and may be assigned in a platform specific manner, or in an extension to this architecture.

In each SMC-R link group, one peer is the server for all TCP connections and the other peer is the client. If there are additional TCP connections between the peers that use SMC-R and have the client and server roles reversed, another SMC-R link group is set up between them with the opposite client-server relationship.

This is required because there are specific responsibilities divided between the client and server in the management of an SMC-R link group.

In this architecture, the decision of whether or not to use an existing SMC-R link group or create a new SMC-R link group for a TCP connection is made exclusively by the server.

Management of the links in an SMC-R link group is also a server responsibility. The server is responsible for adding and deleting links in a link group. The client may request that the server take certain actions but the final responsibility is the server's.

#### 2.2.4. SMC-R link identifiers

This architecture defines multiple identifiers to identify SMC-R links and peers.

- o Link number: This is a one-byte value that identifies an SMC-R link within a link group. Both the server and the client use this number to distinguish an SMC-R link from other links within the same link group. It is only unique within a link group. In order to prevent timing windows that may occur when a server creates a new link while the client is still cleaning up a previously existing link, link numbers cannot be reused until the entire link numbering space has been exhausted.

- o Link User ID: This is an architecturally opaque four byte value that a peer uses to uniquely define an SMC-R link within its own space. This means that a link user ID is unique within one peer only. Each peer defines its own link user ID for a link. The peers exchange this information once during link setup and it is never used architecturally again. The purpose of this identifier is for network management, display, and debugging purposes. For example an operator on a client could provide the operator on the server with the server's link user ID if he requires the server's operator to check on the operation of a link that the client is having trouble with.
- o Peer ID: The SMC-R peer ID uniquely identifies a specific instance of a specific TCP/IP stack. It is required because in clustered and load balancing environments, an IP address does not uniquely identify a TCP/IP stack. An RNIC's MAC/GID also doesn't uniquely or reliably identify a TCP/IP stack because RNICs can go up and down and even be redeployed to other TCP/IP stacks in a multiple partitioned or virtualized environment. The peer ID is not only unique per TCP/IP stack but is also unique per instance of a TCP/IP stack, meaning that if a TCP/IP stack is restarted, its peer ID changes.

### 2.3. SMC-R resilience and load balancing

The SMC-R multi-link architecture provides resilience for network high availability via failover capability to an alternate RoCE adapter.

The SMC-R multilink architecture does not define primary, secondary or alternate roles to the links. Instead there are multiple active links representing multiple redundant RoCE paths over the same LAN.

Assignment of TCP connections to links is unidirectional and asymmetric. This means that the client and server may each choose a separate link for their RDMA writes associated with a specific TCP connection.

If a hardware failure occurs or a QP failure associated with an individual link, then the TCP connections that were associated with the failing link are dynamically and transparently switched to use another available link. The server or the client can detect a

failure and immediately move their TCP connections and then notify their peer via the DELETE LINK LLC command. While the client can notify the server of an apparent link failure with the DELETE LINK LLC command, the server performs the actual link deletion.

The movement of TCP connections to another link can be accomplished with minimal coordination between the peers. The TCP connection movement is also transparent to and non disruptive to the TCP socket application workloads for most failure scenarios. After a failure, the surviving links and all associated hardware must handle the link group's workload.

As each SMC-R peer begins to move active TCP connections to another link all current RDMA write operations must be allowed to complete. Then the moving peer sends a signal to verify receipt of the last successful write by its peer. If this verification fails, the TCP connection must be reset. Once this verification is complete, all writes that failed may then be retried, in order, over the new link. Any data writes or CDC messages for which the sender did not receive write completion must be replayed before any subsequent data or CDC write operations are sent. LLC messages are not retried over the new link because they are dependent on a known link configuration, which has just changed because of the failure. The initiator of an LLC message exchange that fails will be responsible for retrying once the link group configuration stabilizes.

When a new link becomes available and is re-added to the link group then each peer is free to rebalance its current TCP connections as needed or only assign new TCP connections to the newly added link. Both the server and client are free to manage TCP connections across the link group as needed. TCP connection movement does not have to be stimulated by a link failure.

The SMC-R architecture also defines orderly vs. disorderly failover. The type is communicated in the LLC Delete Link command and is simply a means to indicate that the link has terminated (disorderly) or link termination is imminent (orderly). The orderly link deletion could be initiated via operator command or programmatically to bring down an idle link. For example an operator command could initiate orderly shut down of an adapter for service. Implementation of the two types is based on implementation requirements and is beyond the scope of the SMC-R architecture.

### 3. SMC-R Rendezvous architecture

Rendezvous is the process that SMC-R capable peers use to dynamically discover each others' capabilities, negotiate SMC-R connections, set up SMC-R links and link groups, and manage those link groups. A key aspect of SMC-R rendezvous is that it occurs dynamically and automatically, without requiring SMC link configuration to be defined by an administrator.

SMC-R Rendezvous starts with the TCP/IP three-way handshake during which connection peers use TCP options to announce their SMC-R capabilities. If both endpoints are SMC-R capable, then Connection Layer Control (CLC) messages are exchanged between the peers' SMC-R layers over the newly established TCP connection to negotiate SMC-R credentials. The CLC message mechanism is analogous to the messages exchanged by SSL for its handshake processing.

If a new SMC-R link is being set up, Link Layer Control (LLC) messages are used to confirm RDMA connectivity. LLC messages are also used by the SMC-R layers at each peer to manage the links and link groups.

Once an SMC-R link is set up or agreed to by the peers, the TCP sockets are passed to the peer applications which use them as normal. The SMC-R layer, which resides under the sockets layer, transmits the socket data between peers over RDMA using the SMC-R protocol, bypassing the TCP/IP stack.

#### 3.1. TCP options

During the TCP/IP three-way handshake, the client and server indicate their support for SMC-R by including experimental TCP option 254 on the three-way handshake flows, in accordance with RFC 6994 "Shared Use of Experimental TCP Options". The ExID value used is the string 'SMCR' in EBCDIC (IBM-1047) encoding (0xE2D4C3D9). This ExID has been registered in the TCP ExIDs registry maintained by IANA.

After completion of the 3-way TCP handshake each peer queries its peer's options. If both peers set the TCP option on the three-way handshake, inline SMC-R negotiation occurs using CLC messages. If neither peer or only one peer set the TCP option, SMC-R cannot be used for the TCP connection, and the TCP connection completes setup using the IP fabric.

### 3.2. Connection Layer Control (CLC) messages

CLC messages are sent as data payload over the IP network using the TCP connection between SMC-R layers at the peers. They are analogous to the messages used to exchange parameters for SSL.

Use of CLC messages is detailed in the following sections. The following list provides a summary of the defined CLC messages and their purposes:

- o SMC PROPOSAL: Sent from the client to propose that this TCP connection is eligible to be moved to SMC-R. The client identifies itself and its subnet to the server and passes the SMC-R elements for a suggested RoCE path via the MAC and GID.
- o SMC ACCEPT: Sent from the server to accept the client's TCP connection SMC proposal. The server responds to the client's proposal by identifying itself to the client and passing the elements of a RoCE path that the client can use to to perform RDMA writes to the server. This consists of SMC-R link elements such as RoCE MAC, GID, RMB information etc.
- o SMC CONFIRM: Sent from the client to confirm the server's acceptance of SMC connection. The client responds to the server's acceptance by passing the elements of a RoCE path that the server can use to to perform RDMA writes to the client. This consists of SMC-R link elements such as RoCE MAC, GID, RMB information etc.
- o SMC DECLINE: Sent from either the server or the client to reject the SMC connection, indicating the reason the peer must decline the SMC proposal and allowing the TCP connection to revert back to IP connectivity.

### 3.3. LLC messages

Link Layer Control (LLC) messages are sent between peer SMC-R layers over an SMC-R link to manage the link or the link group. LLC messages are sent using RoCE sendmsg with inline data and are 44 bytes long. The 44 bytes size is based on what can fit into a RoCE Work Queue Element (WQE) without requiring the posting of receive buffers.

LLC messages generally follow a request-reply semantic. Each message has a request flavor and a reply flavor, and each request must be confirmed with a reply, except where otherwise noted. Use of LLC messages is detailed in the following sections. The following list provides a summary of the defined LLC messages and their purposes:

- o ADD LINK: Add a new link to a link group. Sent from the server to the client to initiate addition of a new link to the link group, or from the client to the server to request that the server initiate addition of a new link.
- o ADD LINK CONTINUATION: This is a continuation of ADD link that allows the ADD link to span multiple commands, because all the link information cannot be contained in a single ADD LINK message
- o CONFIRM LINK: Used to confirm that RoCE connectivity over a newly created SMC-R link is working correctly. Initiated by the server, and both this message and its reply must flow over the SMC-R link being confirmed.
- o DELETE LINK: When initiated by the server, deletes a specific link from the link group or deletes the entire link group. When initiated by the client, requests that the server delete a specific link or the entire link group.
- o CONFIRM RKEY: Informs the peer on the SMC-R link of the addition of an RMB to the link group.
- o CONFIRM RKEY CONTINUATION: This is a continuation of CONFIRM RKEY that allows the ADD link to span multiple commands, in the event that all of the information cannot be contained in a single CONFIRM RKEY message.
- o DELETE RKEY: Informs the peer on the SMC-R link of the deletion of one or more RMBs from the link group
- o TEST LINK: Verifies that an already-active SMC-R link is active and healthy
- o Optional LLC message: Any LLC message in which the two high order bits of the opcode are b'10' is an optional message and must be silently discarded by a receiving peer that does not support the opcode. No such messages are defined in this version of the architecture, however the concept is defined to allow for toleration of possible advanced, optional functions.

CONFIRM LINK and TEST LINK are sensitive to which link they flow on and must flow on the link being confirmed or tested. The other flows may flow over any active link in the link group. When there are multiple links in a link group, a response to an LLC message must flow over the same link that the original message flowed over, with the following exceptions:

- o ADD LINK request from a server in response to an ADD LINK from a client
- o DELETE LINK request from a server in response to a DELETE LINK from a client

### 3.4. CDC Messages

Connection Data Control (CDC) messages are sent over the RoCE fabric between peers using RoCE sendmsg with inline data, and are 44 bytes long which is based on the size that can fit into a RoCE Work Queue Element (WQE) without requiring the posting of receive buffers. CDC messages are used to describe the socket application data passed via RDMA write operations, and TCP connection state information including producer and consumer cursors, RMBE state information, and failover data validation.

### 3.5. Rendezvous flows

Rendezvous information for SMC-R is exchanged as TCP options on the TCP 3-way handshake flows to indicate capability, followed by in-line TCP negotiation messages to actually do the SMC-R setup. Formats of all rendezvous options and messages discussed in this section are detailed in Appendix A.

#### 3.5.1. First contact

First contact between RoCE peers occurs when a new SMC-R link group is being set up. This could be because no SMC-R links already exist between the peers, or the server decides to create a new SMC-R link group in parallel with an existing one.

##### 3.5.1.1. TCP Options pre-negotiation

The client and server indicate their SMC-R capability to each other using TCP option 254 on the TCP 3-way handshake flows.

A client who wishes to do SMC-R will include TCP option 254 using an ExID equal to the EBCDIC (codepage IBM-1047) encoding of "SMCR" on its SYN flow.

A server that supports SMC-R will include TCP option 254 with the ExID value of EBCDIC "SMCR" on its SYN-ACK flow. Because the server is listening for connections and does not know where client connections will come from, the server implementation may choose to

unconditionally include this TCP option if it supports SMC-R. This may be required for server implementations where extensions to the TCP stack are not practical. For server implementations which can add code to examine and react to packets during the three-way handshake, the server should only include the SMC-R TCP option on SYN-ACK if the client included it on its SYN packet.

A client who supports SMC-R and meets the three conditions outlined above may optionally include the TCP option for SMC-R on its ACK flow, regardless of whether or not the server included it on its SYN-ACK flow. Some TCP/IP stacks may have to include it if the SMC-R layer cannot modify the options on the socket until the 3-way handshake completes. Proprietary servers should not include this option on the ACK flow, since including it on the SYN flow was sufficient to indicate the client's capabilities.

Once the initial three-way TCP handshake is completed, each peer examines the socket options. SMC-R implementations may do this by examining what was actually provided on the SYN and SYN-ACK packets or by performing a `getsockopt()` operation to determine the options set by the peer. If neither peer, or only one peer, specified the TCP option for SMC-R, then SMC-R cannot be used on this connection and it proceeds using normal IP flows and processing.

If both peers specified the TCP option for SMC-R, then the TCP connection is not started yet and the peers proceed to SMC-R negotiation using inline data flows. The socket is not yet turned over to the applications; instead the respective SMC layers exchange CLC messages over the newly formed TCP connection.

#### 3.5.1.2. Client Proposal

If SMC-R is supported by both peers, the client sends an SMC Proposal CLC message to the server. On this flow from client to server it is not immediately apparent if this is a new or existing SMC-R link because in clustered environments a single IP address may represent multiple hosts. This type of cluster virtual IP address can be owned by a network based or host based layer 4 load balancer that distributes incoming TCP connections across a cluster of servers/hosts. Other clustered environments may also support the movement of a virtual IP address dynamically from one host in the cluster to another for high availability purposes. In summary, the client can not pre-determine that a connection is targeting the same host simply by matching the destination IP address for outgoing TCP connections. Therefore it cannot pre-determine the SMC-R link that will be used for a new TCP connection. This information will be

dynamically learned and the appropriate actions will be taken as the SMC-R negotiation handshake unfolds.

On the SMC-R proposal message, the initiator (client) proposes use of SMC-R by including its peer ID and GID and MAC addresses, as well as the IP subnet number of the outgoing interface (if IPv4) or the IP prefix list for the network that the proposal is sent over (if IPv6). At this point in the flow, the client makes no local commitments of resources for SMC-R.

When the server receives the SMC Proposal CLC message, it uses the peer ID provided by the client plus subnet or prefix information provided by the client, to determine if it already has a usable SMC-R link with this SMC-R peer. If there is one or more existing SMC-R links with this SMC-R peer, the server then decides which SMC link it will use for this TCP connection. See subsequent sections for the cases of reusing an existing SMC-R link or creating a parallel SMC link group between SMC-R peers.

If this is a first contact between SMC-R peers the server must validate that it is on the same LAN as the client before continuing. For IPv4, the server does this by verifying that it has an interface with an IP subnet number that matches the subnet number set by the client on the SMC Proposal. For IPv6 it does this by verifying that it is directly attached to at least one IP prefix that was listed by the client in its SMC Proposal message.

If server agrees to use SMC-R, the server begins setup of a new SMC-R link by allocating local QP and RMB resources (setting its QP state to INIT) and providing its full SMC-R information in an SMC Accept CLC message to the client over the TCP connection, along with a flag set indicating that this is a first contact flow. While the SMC Accept message could flow over any route back to the client depending upon IP routing, the SMC-R credentials provided must be for the common subnet or prefix between the server and client, as determined above. If the server cannot or does not want to do SMC-R with the client it sends an SMC Decline CLC message to the client and the connection data may begin flowing using normal TCP/IP flows.

#### 3.5.1.3. Server acceptance

When the client receives the SMC Accept from the server, it uses the combination of the first contact flag, its GID/MAC and the GID/MAC returned by the server plus the LAN that the connection is setting up over and the QP number provided by the server to determine if this is a new or existing SMC-R link.

If it is an existing SMC-R link, and the client agrees to use that link for the TCP connection, see 3.5.2. Subsequent contact below. If it is a new SMC-R link between peers that already have an SMC link, then the server is starting a new SMC link group.

Assuming this is either a first contact between peers or the server is starting a new SMC link group, the client now allocates local QP and RMB resources for the SMC-R link (setting the QP state to RTR or "ready to receive"), associates them with the server QP as learned on the SMC Accept CLC message, and sends an SMC Confirm CLC message to the server over the TCP connection with its SMC-R link information included. The client also starts a timer to wait for the server to confirm the reliable connected QP as described below.

#### 3.5.1.4. Client confirmation

Upon receipt of the client's SMC Confirm CLC message, the server associates its QP for this SMC-R link with the client's QP as learned on the SMC Confirm CLC message and sets its QP state to RTS (ready to send). Now the client and the server have reliable connected QPs.

#### 3.5.1.5. Link (QP) confirmation

Since setting up the SMC-R link and its QPs did not require any network flows on the RoCE fabric, the client and server must now confirm connectivity over the RoCE fabric. To accomplish this, the server will send a "Confirm Link" Link Layer Control (LLC) message to the client over the RoCE fabric. The "Confirm Link" LLC message will provide the server's MAC, GID, and QP information for the connection, allow each partner to communicate the maximum number of links it can tolerate in this link group (the "link limit"), and will additionally provide two link IDs:

- o a one-byte server-assigned Link number that is used by both peers to identify the link within the link group and is only unique within a link group.
- o a four byte link user id. This opaque value is assigned by the server for the server's local use and is provided to the client for management purposes, for example to use in network management displays and products.

When the server sends this message, it will set a timer for receiving confirmation from the client.

When the client receives the server's confirmation "Confirm Link" LLC message it will cancel the confirmation timer it set when it sent the

SMC Confirm message. It will also advance its QP state to RTS and respond over the RoCE fabric with a "Confirm Link" response LLC message, providing its MAC, GID, QP number, link limit, confirming the one byte link number sent by the server, and providing its own four byte link user id to the server.

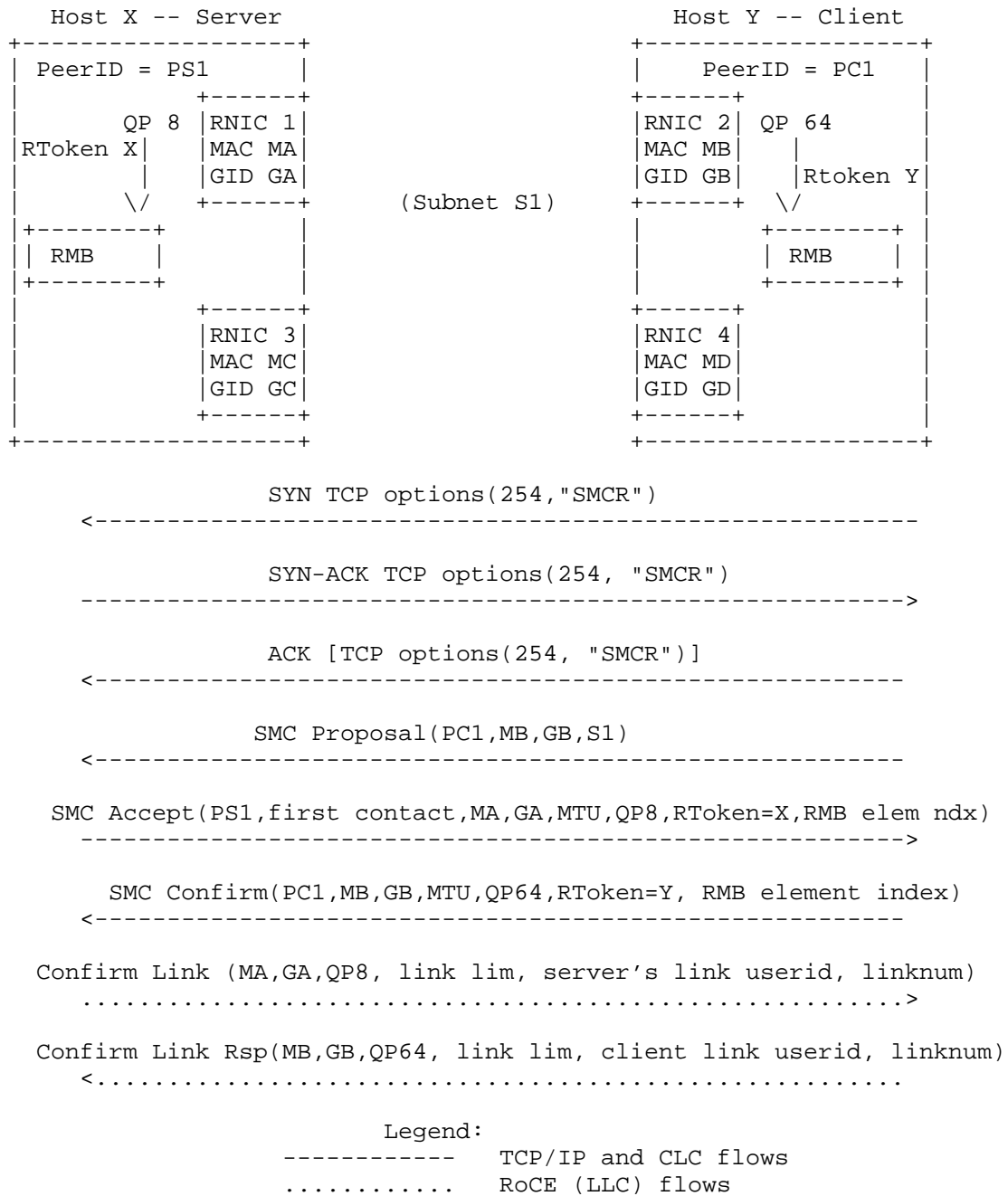


Figure 8 First contact rendezvous flows

Technically, the data for the TCP connection could now flow over the RoCE path. However if this is first contact, there is no alternate for this recently established RoCE path. Since in the current architecture there is no failover from RoCE to IP once connection data starts flowing, this means that a failure of this path would disrupt the TCP connection, meaning that the level of redundancy and failover is less than that provided by IP. If the network has alternate RoCE paths available, they would not be usable at this point, which is an unacceptable condition

#### 3.5.1.6. Second SMC-R link setup

Because of the unacceptable situation described above, TCP data will not be allowed to flow on the newly established SMC-R link until a second path has been set up, or at least attempted.

If the server has a second RNIC available on the same LAN, it attempts to set up the second SMC-R link over that second RNIC. If it only has one RNIC available on the LAN, it will attempt to set up the second SMC-R link over that one RNIC. In the latter case, the server is attempting to set up an asymmetric link, in case the client does have a second RNIC on the LAN.

In either case the server allocates a new QP over the RNIC it is attempting to use for the second link, assigns a link number to the new link and also creates an RToken for the RMB over this second QP (note that this means that the first and second QP each has its own RToken to represent the same RMB). The server provides this information, as well as the MAC and GID of the RNIC it is attempting set up the second link over in an "Add Link" LLC message which it sends to the client over the SMC-R link that is already set up.

##### 3.5.1.6.1. Client processing of "Add Link" LLC message from server

When the client receives the server's "Add Link" LLC message, it examines the GID and MAC provided by the server to determine if the server is attempting to use the same server-side RNIC as the existing SMC-R link, or a different one.

If the server is attempting to use the same server-side RNIC as the existing SMC-R link, then the client verifies that it has a second RNIC on the same LAN. If it does not, the client rejects the "Add Link" request from the server, because the resulting link would be a parallel link which is not supported within a link group. If the client does have a second RNIC on the same LAN, it accepts the request and an asymmetric link will be set up.

If the server is using a different server-side RNIC from the existing SMC-R link then the client will accept the request and a second SMC-R link will set up in this SMC-R link group. If the client has a second RNIC on the same LAN, that second RNIC will be used for the second SMC-R link, creating symmetric links. If the client does not have a second RNIC on the same LAN, it will use the same RNIC as was used for the initial SMC-R link, resulting in the setup of an asymmetric link in the SMC-R link group.

In either case, when the client accepts the server's "Add Link" request, it allocates a new QP on the chosen RNIC and creates an Rkey over that new QP for the client-side RMB for the SMC link group, then sends an "Add Link" reply LLC message to the server providing that information as well as echoing the Link number that was set by the server.

If the client rejects the server's "Add Link" request, it sends an "Add Link" reply LLC message to the server with the reason code for the rejection.

#### 3.5.1.6.2. Server processing of "Add Link" reply LLC message from the client

If the client sends a negative response to the server or no reply is received, the server frees the RoCE resources it had allocated for the new link. Having a single link in an SMC-R link group is undesirable and the server's recovery is detailed in C.8. Failure to add second SMC-R link to a link group.

If the client sends a positive reply to the server with MAC/GID/QP/Rkey information, the server associates its QP for the new SMC-R link to the QP that the client provided. Now the new SMC-R link is in the same situation that the first was in after the client sent its ACK packet - there is a reliable connected QP over the new RoCE path, but there have been no RoCE flows to confirm that it's actually usable. So at this point the client and server will exchange "Confirm Link" LLC messages just like they did on the first SMC-R link.

If either peer receives a failure during this second "Confirm Link" LLC exchange (either an immediate failure which implies that the message did not reach the partner, or a timeout), it sends a "Delete Link" LLC message to the partner over the first (and now only) link in the link group which must be acknowledged before data can flow on the single link in the link group.



#### 3.5.1.6.3. Exchange of Rkeys on second SMC-R link

Note that in the scenario described here, first contact, there is only one RMB Rkey to exchange on the second SMC-R link and it is exchanged in the Add Link Continuation request and reply. In scenarios other than first contact, for example, adding a new SMC-R link to a longstanding link group with multiple RMBs, additional flows will be required to exchange additional RMB Rkeys. See 3.5.5.2.3. Adding a new SMC-R link to a link group with multiple RMBs for more details on these flows

#### 3.5.1.6.4. Aborting SMC-R and falling back to IP

If both partners don't provide the SMC-R TCP option during the 3 way TCP handshake, the connection falls back to normal TCP/IP. During the SMC-R negotiation that occurs after the 3 way TCP handshake, either partner may break off SMC-R by sending an SMC Decline CLC message. The SMC Decline CLC message may be sent in place of any expected message, and may also be sent during the Confirm Link LLC exchange if there is a failure before any application data has flowed over the RoCE fabric. For more detail on exactly when an SMC Decline can flow during link group setup, see C.1. SMC Decline during CLC negotiation and C.2. SMC Decline during LLC negotiation

If this fallback to IP happens while setting up a new SMC-R link group, the RoCE resources allocated for this SMC-R link group relationship are torn down and it will be retried as a new SMC-R link group next time a connection starts between these peers with SMC-R proposed. Note that if this happens because one side doesn't support SMC-R, there will be very little to tear down as the TCP option will have failed to flow either on the initial SYN or the SYN-ACK, before either side had reserved any local RoCE resources.

#### 3.5.2. Subsequent contact

"Subsequent contact" means setting up a new TCP connection between two peers that already have an SMC-R link group between them, and reusing the existing SMC-R link group. In this case it is not necessary to allocate new QPs. However it is possible that a new RMB has been allocated for this TCP connection, if the previous TCP connection used the last element available in the previously used RMB, or for any other implementation-dependent reason. For this reason, and for convenience and error checking, the same TCP option 254 followed by inline negotiation method described for initial contact will be used for subsequent contact, but the processing differs in some ways. That processing is described below.

### 3.5.2.1. SMC-R proposal

When the client begins the inline negotiation with the server, it does not know if this is a first contact or a subsequent contact. The client cannot know this information until it sees the server's peer ID to determine whether or not it already has an SMC-R link with this peer that it can use. There are several reasons why it is not sufficient to use the partner IP address, subnet, VLAN or other IP information to make this determination. The most obvious reason is distributed systems: if the server IP address is actually a virtual IP address representing a distributed cluster, the actual host serving this TCP connection may not be the same as the host that served the last TCP connection to this same IP address.

After the TCP three way handshake, assuming both partners indicate SMC-R capability, the client builds and sends the SMC Proposal CLC message to the server in exactly the same manner as it does in the first contact case, and in fact at this point doesn't know if it's first contact or subsequent contact. As in the first contact case, the client sends its Peer ID value, suggested RNIC GID/MAC, and IP subnet or prefix information.

Upon receiving the client's proposal, the server looks up the peer ID provided to determine if it already has a usable SMC-R link group with this peer. If it does already have a usable SMC-R link group, the server then needs to decide if it will use the existing SMC-R link group, or create a new link group. For the new link group case, see 3.5.3. First contact variation: creating a parallel link group, below.

For this discussion assume the server decides to use the existing SMC-R link group for the TCP connection, which is expected to be the most common case. The server is responsible for making this decision. Then the server needs to communicate that information to the client, but it is not necessary to allocate, associate, and confirm QPs for the chosen SMC-R link. All that remains to be done is to set up RMB space for this TCP connection.

If one of the RMBs already in use for this SMC-R link group has an available element that uses the appropriate buffer size, the server merely chooses one for this TCP connection and then sends an SMC Accept CLC message, providing the full RoCE information for the chosen SMC-R link to the client, using the same format as the SMC Accept CLC message described in the initial contact section above.

The server may choose to use the SMC-R link that matches the suggested MAC/GID provided by the client on the SMC Proposal for its

RDMA writes but is not obligated to. The final decision on which specific SMC-R link to assign a TCP connection to is an independent server and client decision.

It may be necessary for the server to allocate a new RMB for this connection. The reasons for this are implementation dependent and could include: no available space in existing RMB or RMBs, or desire to allocate a new RMB that uses a different buffer size from the ones already created, or any other implementation dependent reason. In this case the server will allocate the new RMB and then perform the flows described in 3.5.5.2.1. Adding a new RMB to an SMC-R link group. Once that processing is complete, the server then provides the full RoCE information, including the new Rkey, for this connection on an SMC Confirm CLC message to the client.

#### 3.5.2.2. SMC-R acceptance

Upon receiving the SMC Accept CLC message from the server, the client examines the RoCE information provided by the server to determine if this is a first contact for a new SMC link group, or subsequent contact for an existing SMC-R link group. It is subsequent contact if the server side peer ID, GID, MAC and QP number provided on the packet match a known SMC-R link, and the "first contact" flag is not set. If this is not the case, for example the GID and MAC match but the QP is new, then the server is creating a new, parallel SMC-R link group and this is treated as a first contact.

A different RMB RToken does not indicate a first contact as the server may have allocated a new RMB, or be using several RMBs for this SMC-R link. The client needs the server's RMB information only for its RDMA writes to the server, and since there is no requirement for symmetric RMBs, this information is simply control information for the RDMA writes on this SMC-R link.

The client must validate that the RMB element being provided by the server is not in use by another TCP connection on this SMC-R link group. This validation must validate the new <rtoken, index> across all known <rtoken, index> on this link group. See 4.4.2. RMB element reuse and conflict resolution for the case in which the server tries to use an RMB element that is already in use on this link group.

Once the client has determined that this TCP connection is a subsequent contact over an existing SMC link, it performs a similar RMB allocation process as the server did: it either allocates an element from an RMB already associated with this SMC-R link, or it allocates a new RMB and associates it with this SMC-R link and then chooses an element out of it.

If the client allocates a new RMB for this TCP connection, it performs the processing described in 3.5.5.2.1. Adding a new RMB to an SMC-R link group. Once that processing is complete, the client provides its full RoCE information for this TCP connection on an SMC Confirm CLC message.

Because an SMC-R link with a verified connected QP already exists and is being reused, there is no need for verification or alternate QP selection flows or timers.

#### 3.5.2.3. SMC-R confirmation

When the server receives the client's SMC Confirm CLC message on a subsequent contact, it verifies the following:

- o the RMB element provided by the client is not already in use by another TCP connection on this SMC-R link group (see section 4.4.2. RMB element reuse and conflict resolution for the case in which it is).
- o The MAC/GID/QP info provided by the client matches an active link within the link group. The client is free to select any valid / active link. The client is not required to select the same link as the server.

If this validation passes, the server stores the client's RMB information for this connection and the RoCE setup of the TCP connection is complete.

#### 3.5.2.4. TCP data flow race with SMC Confirm CLC message

On a subsequent contact TCP/IP connection, a peer may send data as soon as it has received the peer RMB information for the connection. There are no additional RoCE confirmation flows, since the QPs on the SMC link are already reliably connected and verified.

In the majority of cases the first data will flow from the client to the server. The client must send the SMC Confirm CLC message before sending any connection data over the chosen SMC-R link, however the client need not wait for confirmation of this message, and in fact there will be no such confirmation. Since the server is required to have the RMB fully set up and ready to receive data from the client before sending SMC Accept CLC message, the client can begin sending data over the SMC-R link immediately upon completing the send of the SMC Confirm CLC message.

It is possible that data from the client will arrive into the server side RMB before the SMC Confirm CLC message from the client has been processed. In this case the server must handle this race condition, and not provide the arrived TCP data to the socket application until the SMC Confirm CLC message has been received and fully processed, opening the socket.

If the server has initial data to send to the client which is not a response to the client (this case should be rare), it can send the data immediately upon receiving and processing the SMC Confirm CLC message from the client. The client must have opened the TCP socket to the client application upon sending of SMC Confirm CLC message so the client will be ready to process data from the server.

### 3.5.3. First contact variation: creating a parallel link group

Recall that parallel SMC-R links within an SMC-R link group are not supported. These are multiple SMC-R links within a link group that use the same network path. However, multiple SMC-R link groups between the same peers are supported. This means that if multiple SMC-R links over the same RoCE path are desired, it is necessary to use multiple SMC-R link groups. While not a recommended practice, this could be done for platform specific reasons, like QP separation of different workloads. Only the server can drive the creation of multiple SMC-R link groups between peers.

At a high level, when the server decides to create an additional SMC-R link group with a client it already has an SMC-R link group with, the flows are basically the same as the normal "first contact" case described above. The following provides more detail and clarification of processing in this case.

When the server receives the SMC Proposal CLC message from the client and using the GID/MAC info determines that it already has an SMC-R link group with this client, the server can either reuse the existing SMC-R link group (detailed in 3.5.2. Subsequent contact above) or it can create a new SMC-R link group in addition to the existing one.

If the server decides to create a new SMC-R link group, it does the same processing it would have done for first contact: allocate QP and RMB resources as well as alternate QP resources, and communicate the QP and RMB information to the client on the SMC Accept CLC message with the "first contact" flag set.

When the client receives the server's SMC Accept CLC message with the new QP information and the "first contact" flag, it knows the server is creating a new SMC-R link group even though it already has an SMC-

R link group with the server. In this case the client will also allocate a new QP for this new SMC link and allocate an RMB for this link and generate an Rkey for it.

Note that multiple SMC-R link groups between the same peers must access different RMB resources, so new RMBs will be required. Using the same RMBs that are in use in another SMC-R link group is not permitted.

The client then associates its new QP with the server's new QP and sends its SMC Confirm CLC message back to the server providing the new QP/RMB information and sets its confirmation timer for the new SMC-R link.

When the server receives the client's SMC Confirm CLC message it associates its QP with the client's QP as learned on the SMC Confirm CLC message and sends a confirmation LLC message. The rest of the flow, with the confirmation QP and setup of additional SMC-R links, unfolds just like the first contact case.

#### 3.5.4. Normal SMC-R link termination

The normal sockets API trigger points are used by the SMC-R layer to initiate SMC-R connection termination flows. The main design point for SMC-R normal connection flows is to use the SMC-R protocol to first shutdown the SMC-R connection and free up any SMC-R RDMA resources and then allow the normal TCP connection termination protocol (i.e. FIN processing) to drive cleanup of the TCP connection that exists on the IP fabric. This design point is very important in ensuring that RDMA resources such as the RMBEs are only freed and reused when both SMC-R end points are completely done with their RDMA Write operations to the partner's RMBE.

When the last TCP connection over an SMC-R link group terminates, the link group can be terminated. Similar to creation of SMC-R links and link groups, the primary responsibility for determining that normal termination is needed and initiating it lies with the server. Implementations may opt to set timers to keep SMC-R link groups up for a specified time after the last TCP connection ends, to avoid churn in cases when TCP connections come and go regularly.

The link or link group may also be terminated as a result of an operator initiated command. This command can be entered at either the client or the server. If entered at the client, the client requests that the server perform link or link group termination, and the responsibility for doing so ultimately lies with the server.

When the server determines that the SMC-R link group is to be terminated, it sends a DELETE LINK LLC message to the client, with a flag set indicating that all links in the link group are to be terminated. After receiving confirmation from the adapter that the DELETE LINK LLC message has been sent, the server can clean up its end of the link group (QPs, RMBs, etc). Upon receipt of the DELETE LINK message from the server, the client must immediately comply and clean up its end of the link group. Any TCP connections that the client believes to be active on the link group must be immediately terminated.

The client can request that the server delete the link group as well. The client does this by sending a DELETE LINK message to the server indicating that cleanup of all links is requested. The server must comply by sending a DELETE LINK to the client and processing as described above. If there are TCP connections active on the link group when the server receives this request, they are immediately terminated by sending a RST flow over the IP fabric.

### 3.5.5. Link group management flows

#### 3.5.5.1. Adding and deleting links in an SMC-R link group

The server has the lead role in managing the composition of the link group. Links are added to link group by the server. The client may notify the server of new conditions that may result in the server adding a new link, but the server is ultimately responsible. In general links are deleted from the link group by the server, however in certain error cases the client may inform the server that a link must be deleted and treat it as deleted without waiting for action from the server. These flows are detailed in the following sections

##### 3.5.5.1.1. Server initiated Add Link processing

As described in previous sections, the server initiates an Add Link exchange to create redundancy in a newly created link group. Once a link group is established the server may also initiate Add Link for other reasons, including:

- o Availability of additional resources on the server host to support an additional SMC-R link. This may include the provisioning of an additional RNIC, more storage becoming available to support additional QP resources, operator command, or any other implementation dependent reason. Note that, to be available for an existing link group, a new RNIC must be attached to the same RoCE LAN that the link group is using.

- o Receipt of notification from the client that additional resources on the client are available to support an additional SMC-R link. See 3.5.5.1.2. Client initiated Add Link processing.

Server initiated Add Link processing in an established SMC-R link group is the same as the Add Link processing described in 3.5.1.6. Second SMC-R link setup with the following changes:

- o If an asymmetric SMC-R link already exists in the link group a second asymmetric link will not be created. Only one asymmetric link is permitted in a link group.
- o TCP data flow on already existing link(s) in the link group is not halted or otherwise affected during the process of setting up the additional link.

In no case will the server initiate Add Link processing if the link group already has the maximum number of links negotiated by the partners.

#### 3.5.5.1.2. Client initiated Add Link processing

If an additional RNIC becomes available for an existing SMC-R link group on the client's side, the client notifies the server by sending an Add Link request LLC message to the server. Unlike an Add Link request sent by the server to the client, this Add Link request merely informs the server that the client has a new RNIC. If the link group lacks redundancy, or has redundancy only on an asymmetric link with a single RNIC on the client side, the server must initiate an Add Link exchange in response to this message, to create or improve the link group's redundancy.

If the link group already has symmetric link redundancy but has fewer than the negotiated maximum number of links, the server may respond by initiating an Add Link exchange to create a new link using the client's new resource but is not required to.

If the link group already has the negotiated maximum number of links, the server must ignore the client's Add Link request LLC message.

Because the server is not required to respond to the client's Add Link LLC message in all cases, the client must not wait for a response or throw an error if one does not come.

#### 3.5.5.1.3. Server initiated Delete Link Processing

Reasons that a server may delete a link include:

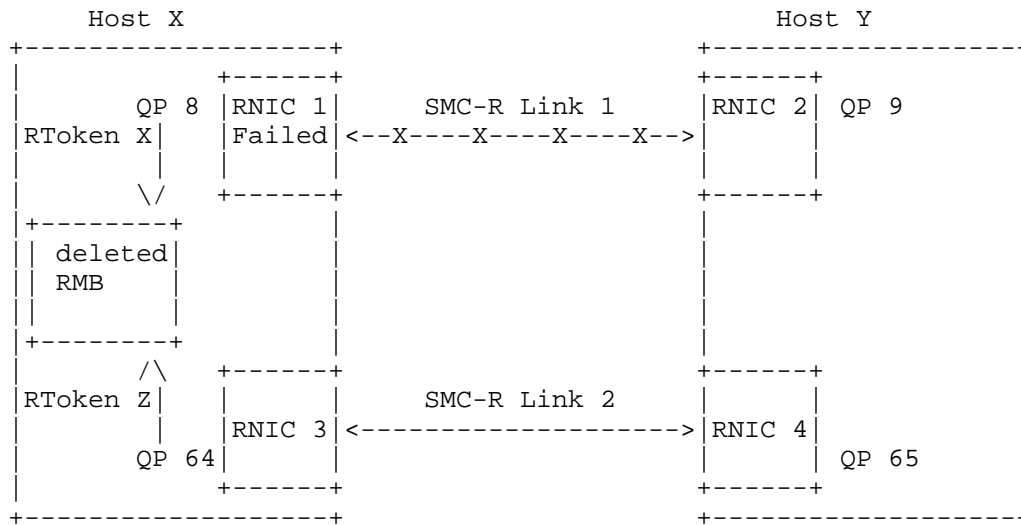
- o The link has not been used for TCP connections for an implementation defined time interval, and deleting the link will not cause the link group to lack redundancy
- o An error in resources supporting the link. These may include but are not limited to: RNIC errors, QP errors, software errors
- o The RNIC supporting this SMC-R link is being taken down, either because of an error case or because of an operator or software command.

If a link being deleted is supporting TCP connections, and there are one or more surviving links in the link group, the TCP connections are moved to the surviving links. For more information on this processing see 2.3. SMC-R resilience and load balancing.

The server deletes a link from the link group by sending a Delete Link request LLC message to the client over any of the usable links in the link group. Because the Delete Link LLC message specifies which link is to be deleted, it may flow over any link in the link group. The server must not clean up its RoCE resources for the link until the client responds.

The client responds to the server's Delete Link request LLC message by sending the server a Delete Link response LLC message. The client must respond positively; it cannot decline to delete the link. Once the server has received the client's Delete Link response, both sides may clean up their resources for the link.

Positive write completion or other indication from the RNIC on the client's side is sufficient to indicate to the client that the server has received the Delete Link response.



```
DELETE LINK(Request, link number = 1,
             .....>
             reason code = RNIC failure)

DELETE LINK(Response, link number = 1)
<.....

(note, architecturally this exchange can flow over either
SMC-R link but most likely flows over link 2 since
the RNIC for link 1 has failed)
```

Figure 10 Server initiated Delete Link flow

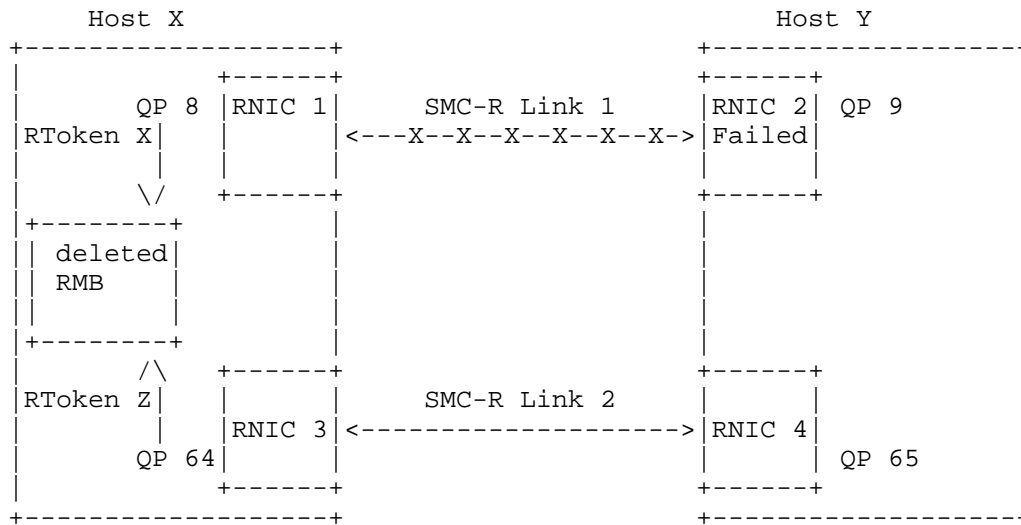
#### 3.5.5.1.4. Client initiated Delete Link request

The client may request that the server delete a link for the same reasons that the server may delete a link, except for inactivity timeout.

Because the client depends on the server to delete links, there are two types of delete requests from client to server:

- o Orderly: the client is requesting that the server delete the link when able. This would result from an operator command to bring down the RNIC or some other nonfatal reason. In this case the server is required to delete the link, but may not do it right away.
- o Disorderly: the server must delete the link right away, because the client has experienced a fatal error with the link.

In either case the server responds by initiating a Delete Link exchange with the client as described in the previous section. The difference between the two is whether the server must do so immediately or can delay for an opportunity to gracefully delete the link.



```
DELETE LINK(Request, link number = 1, disorderly,
<.....
      reason code = RNIC failure)

DELETE LINK(Request, link number = 1,
.....>
      reason code = RNIC failure)

DELETE LINK(Response, link number = 1)
<.....

(note, architecturally this exchange can flow over either
SMC-R link but most likely flows over link 2 since
the RNIC for link 1 has failed)
```

Figure 11 Client-initiated Delete Link

### 3.5.5.2. Managing multiple Rkeys over multiple SMC-R links in a link group

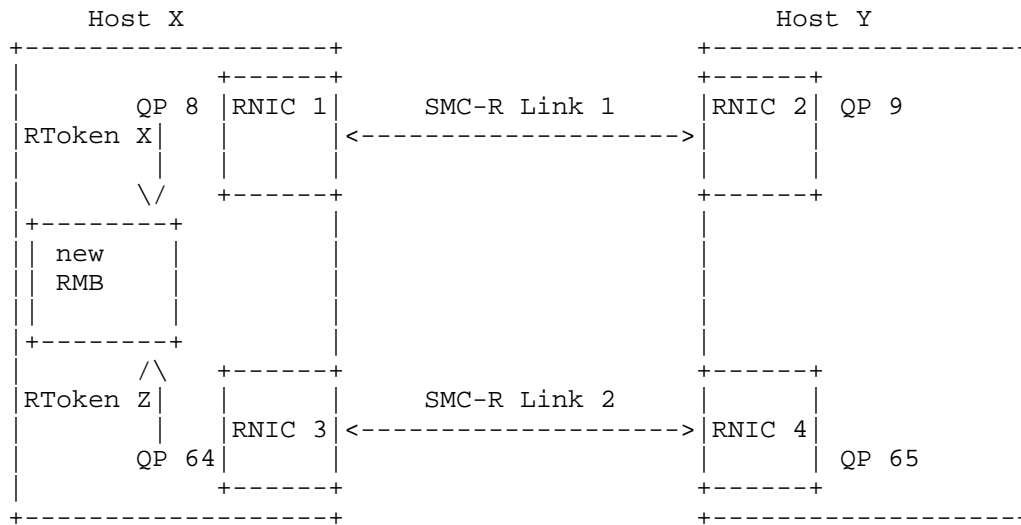
After the initial contact sequence completes and the number of TCP connections increases it is possible that the SMC peers could add additional RMBs to the Link Group. Recall that each peer independently manages its RMBs. Also recall that an RMB's RToken is specific to a QP, which means that when there are multiple SMC-R links in a link group, each RMB accessed with the link group requires a separate RToken for each SMC-R link in the group.

Each RMB that is added to a link must be added to all links within the Link Group. The set of RMBs created for the Link is called the "RToken Set". The RTokens must be exchanged with the peer. As RMBs are added and deleted, the RToken Set must remain in sync.

#### 3.5.5.2.1. Adding a new RMB to an SMC-R link group

A new RMB can be added to an SMC-R link group on either the client or the server side. When an additional RMB is added to an existing SMC-R link group, that RMB must be associated with the QPs for each link in the link group. Therefore when an RMB is added to an SMC-R link group, its RMB RToken for each SMC-R link's QP must be communicated to the peer.

The tokens for a new RMB added to an existing SMC-R link group are communicated using "Confirm Rkey" LLC messages, as shown in Figure 12. The RToken set is specified as pairs: an SMC link number, paired with the new RMB's RToken over that SMC Link. To preserve failover capability, any TCP connection that uses a newly added RMB cannot go active until all RTokens for the RMB have been communicated for all the links in the link group.



```

CONFIRM RKEY(Request, Add,
    .....>
    RToken set((Link 1,RToken X),(Link2,RToken Z)))

CONFIRM RKEY(Response, Add,
    <.....
    RToken set((Link 1,RToken X),(Link2,RToken Z)))

(note, this exchange can flow over either SMC-R link)

```

Figure 12 Add RMB to existing link group

Implementations may choose to proactively add RMBs to link groups in anticipation of need. For example, an implementation may add a new RMB when all of its existing RMBs are over a certain threshold percentage used.

A new RMB may also be added to an existing link group on an as needed basis. For example, when a new TCP connection is added to the link group but there are no available RMB elements. In this case the CLC exchange is paused while the peer that requires the new RMB adds it. An example of this is illustrated in figure 13.

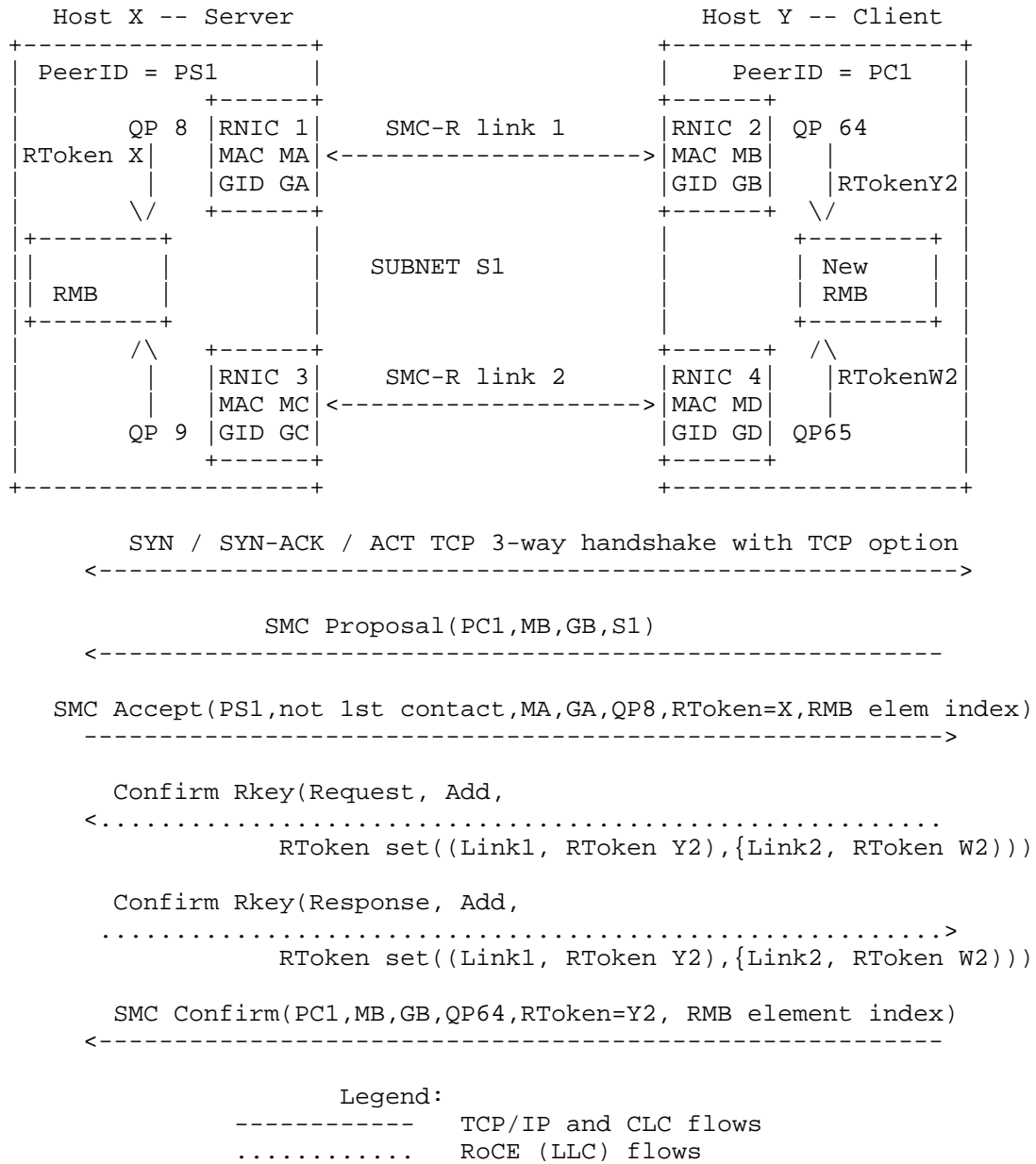
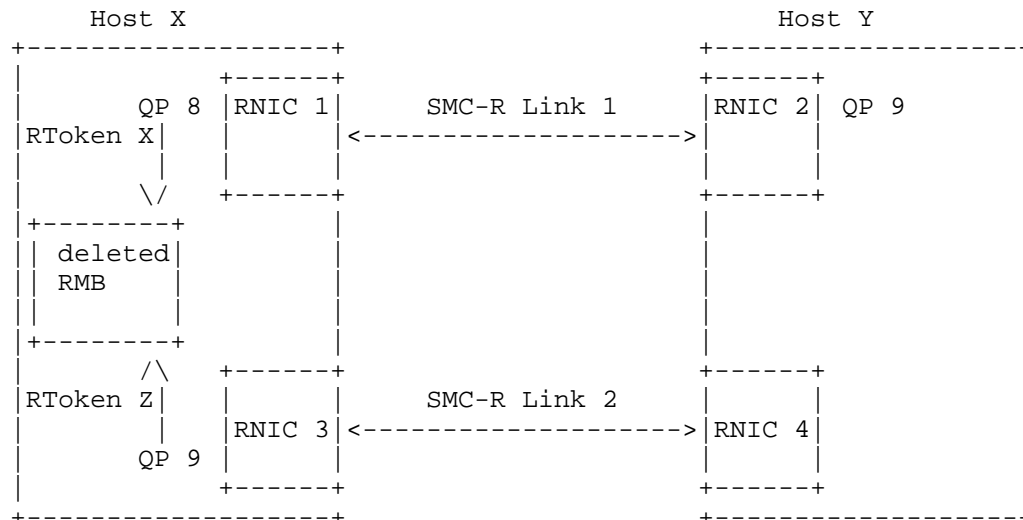


Figure 13 Client adds RMB during TCP connection setup

### 3.5.5.2.2. Deleting an RMB from an SMC-R link group

Either peer can delete one or more of its RMBs as long as it is not being used for any TCP connections. Ideally an SMC-R peer would use a timer to avoid freeing an RMB immediately after the last TCP connection stops using it, to keep the RMB available for later TCP connections and avoid thrashing with addition and deletion of RMBs. Once an SMC-R peer decides to delete an RMB, it sends a DELETE RKEY LLC message to its peer. It can then free the RMB once it receives a response from the peer. Multiple RMBs can be deleted in a DELETE RKEY exchange.

Note that in a DELETE RKEY message, it is not necessary to specify the full RToken for a deleted RMB. The RMB's Rkey over one link in the link group is sufficient to specify which RMB is being deleted.



```
DELETE RKEY(Request, Rkey list(Rkey X))
.....>

DELETE RKEY(Response, Rkey list(Rkey X))
<.....

(note, this exchange can flow over either SMC-R link)
```

Figure 14 Delete RMB from SMC-R link group

### 3.5.5.2.3. Adding a new SMC-R link to a link group with multiple RMBs

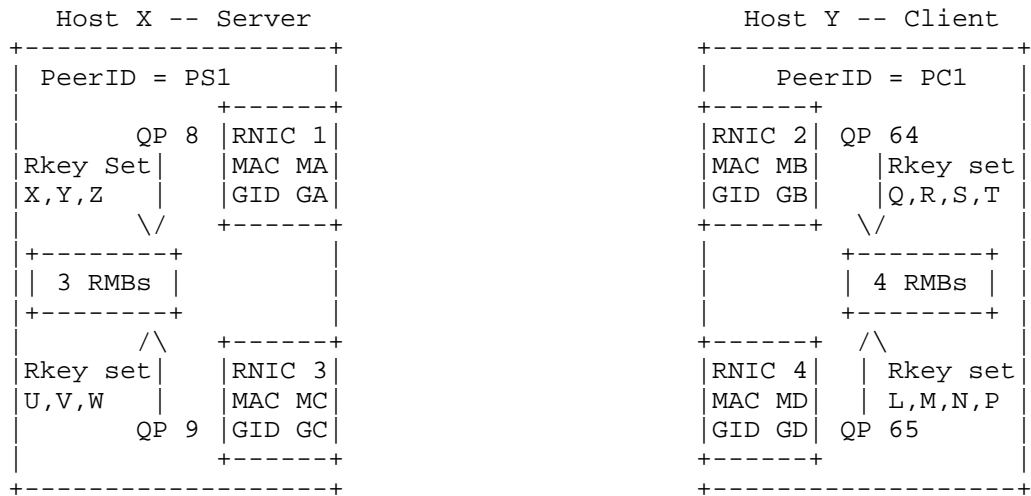
When a new SMC-R link is added to an existing link group, there could be multiple RMBs on each side already associated with the link group. There could also be a different number of RMBs on one side as on the other, because each peer manages its RMBs independently. Each of these RMBs will require a new RToken to be used on the new SMC-R link, and then those new RTokens must be communicated to the peer. This requires two-way communication as the server will have to communicate its RTokens to the client and vice versa.

RTokens are communicated between peers in pairs. Each RToken pair consists of:

- o The RToken for the RMB, as is already known on an existing SMC-R link in the link group
- o The RToken for the same RMB, to be used on the new SMC-R link.

These pairs are required to ensure that each peer knows which RTokens across QPs are equivalent.

The "Add Link" request and response LLC messages do not have room to contain any RToken pairs. "Add Link continuation" LLC messages are used to communicate these pairs, as shown in Figure 15. The "Add Link Continuation" LLC messages are sent on the same SMC-R link that the "Add Link" LLC messages were sent over, and in both the "Add Link" and the "Add Link Continuation" LLC messages, the first RToken in each RToken pair will be the RToken for the RMB as known on the SMC-R link that the LLC message is being sent over.



```

ADD link request (QP9,MC,GC, link number=2)
.....>

ADD link response (QP65,MD,GD, link number=2)
<.....

ADD link continuation req(RToken Pairs=((X,U),(Y,V),(Z,W)))
.....>

ADD link continuation rsp(RToken Pairs=((Q,L),(R,M),(S,N),(T,P)))
<.....

Confirm Link Req/Rsp exchange on link 2
<.....>

```

#### Legend:

```

----- TCP/IP and CLC flows
.....  RoCE (LLC) flows

```

Figure 15 Exchanging Rkeys when a new link is added to a link group

### 3.5.5.3. Serialization of LLC exchanges, and collisions

LLC flows can be divided into two main groups for serializaion considerations.

The first group is LLC messages that are independent and can flow at any time. These are one-time, unsolicited messages that either do

not have a required response, or that have a simple response that does not interfere with the operations of another group of messages. These messages are:

- o TEST LINK from either the client or the server: This message requires a TEST LINK response to be returned, but does not affect the configuration of the link group or the Rkeys.
- o ADD LINK from the client to the server: This message is provided as an "FYI" to the server to let it know that the client has an additional RNIC available. The server is not required to act upon or respond to this message.
- o DELETE\_LINK from the client to the server: This message informs the server that the client has either experienced an error or problem that requires a link or link group to be terminated, or that an operator has commanded that a link or link group be terminated. The server does not respond directly to the message, rather it initiates a DELETE LINK exchange as a result of receiving it.
- o DELETE LINK from the server to the client with the "delete entire link group" flag set: This message informs the client that the entire link group is being deleted.

The second group is LLC messages that are part of an exchange of LLC messages that affects link group configuration that must complete before another exchange of LLC messages that affects link group configuration can be processed. When a peer knows that one of these exchanges is in progress, it must not start another exchange. These exchanges are:

- o ADD LINK / ADD LINK response / ADD LINK CONTINUATION / ADD LINK CONTINUATION response / CONFIRM LINK / CONFIRM LINK RESPONSE: This exchange, by adding a new link, changes the configuration of the link group.
- o DELETE LINK / DELETE LINK response initiated by the server: This exchange, by deleting a link, changes the configuration of the link group.

- o CONFIRM RKEY / CONFIRM RKEY response or DELETE RKEY / DELETE RKEY response: This exchange changes the RMB configuration of the link group. RKeys can not change while links are being added or deleted (while ADD or DELETE LINK is in progress). However, CONFIRM RKEY and DELETE RKEY are unique in that both the client and server can independently manage (add or remove) their own RMBs. This allows each peer to concurrently change their RKeys and therefore concurrently send CONFIRM RKEY or DELETE RKEY requests. The concurrent CONFIRM RKEY or DELETE RKEY requests can be independently processed and do not represent a collision

Because the server is in control of the configuration of the link group, many timing windows and collisions are avoided but there are still some that must be handled.

#### 3.5.5.3.1. Collisions with ADD LINK / CONFIRM LINK exchange

Colliding LLC message: TEST LINK

Action to resolve: Send immediate TEST LINK reply

Colliding LLC Message: ADD LINK from client to server

Action to resolve: Server ignores the ADD LINK message. When client receives server's ADD LINK, client will consider that message to be in response to its ADD LINK message and the flow works. Since both client and server know not to start this exchange if an ADD LINK operation is already underway, this can only occur if the client sends this message before receiving the server's ADD LINK and this message crosses with the server's ADD LINK message, therefore the server's ADD LINK arrives at the client immediately after the client sent this message.

Colliding LLC Message: DELETE LINK from client to server, specific link specified

Action to resolve: Server queues the DELETE link message and processes after the ADD LINK exchange completes. If it is an orderly link termination, it can wait until after this exchange continues. If it is disorderly and the link affected is the one that the current exchange is using, the server will discover the outage when a message in this exchange fails.

Colliding LLC Message: DELETE LINK from client to server, entire link group to be deleted

Action to resolve: Immediately clean up the link group

Colliding LLC message: CONFIRM RKEY from the client

Action to resolve: Send negative CONFIRM\_RKEY response to the client. Once the current exchange finishes, client will have to recompute its Rkey set to include the new link, and start a new CONFIRM RKEY exchange.

#### 3.5.5.3.2. Collisions during DELETE LINK exchange

Colliding LLC Message: TEST LINK from either peer

Action to resolve: Send immediate TEST LINK response

Colliding LLC message: ADD LNK from client to server

Action to resolve: Server queues the ADD LINK and processes it after the current exchange completes

Colliding LLC message: DELETE LINK from client to server (specific link)

Action to resolve: Server queues the DELETE link message and processes after the current exchange completes. If it is an orderly link termination, it can wait until after this exchange continues. If it is disorderly and the link affected is the one that the current exchange is using, the server will discover the outage when a message in this exchange fails

Colliding LLC message: DELETE LINK from either client or server, deleting the entire link group

Action to resolve: immediately clean up the link group

Colliding LLC message: CONFIRM\_RKEY from client to server

Action to resolve: Send negative CONFIRM\_RKEY response to the client. Once the current exchange finishes, client will have to recompute its Rkey set to include the new link, and start a new CONFIRM RKEY exchange

#### 3.5.5.3.3. Collisions during CONFIRM\_RKEY exchange

Colliding LLC Message: TEST LINK

Action to resolve: Send immediate TEST LINK reply

Colliding LLC message: ADD LINK from client to server

Action to resolve: Queue the ADD LINK and process it after the current exchange completes

Colliding LLC message: ADD LINK from server to client (CONFIRM RKEY exchange was initiated by the client and it crossed with the server initiating an ADD LINK exchange)

Action to resolve: Process the ADD LINK. Client will receive a negative CONFIRM RKEY from the server and will have to redo this CONFIRM RKEY exchange after the ADD LINK exchange completes.

Colliding LLC message: DELETE LINK from client to server, specific link to be deleted (CONFIRM RKEY exchange was initiated by the server and it crossed with the client's DELETE LINK request)

Action to resolve: Server queues the DELETE link message and processes after the ADD LINK exchange completes. If it is an orderly link termination, it can wait until after this exchange continues. If it is disorderly and the link affected is the one that the current exchange is using, the server will discover the outage when a message in this exchange fails.

Colliding LLC message: DELETE LINK from server to client, specific link deleted (CONFIRM RKEY exchange was initiated by the client and it crossed with the server's DELETE LINK)

Action to resolve: Process the DELETE LINK. Client will receive a negative CONFIRM RKEY from the server and will have to redo this CONFIRM RKEY exchange after the ADD LINK exchange completes.

Colliding LLC message: DELETE LINK from either client or server, entire link group deleted

Action to resolve: immediately clean up the link group

Colliding LLC message: CONFIRM LINK from the peer that did not start the current CONFIRM LINK exchange

Action to resolve: Queue the request and process it after the current exchange completes.

#### 4. SMC-R memory sharing architecture

##### 4.1. RMB element allocation considerations

Each TCP connection using SMC-R must be allocated a RMBE by each SMC-R peer. This allocation is performed by each end point independently to allow each end point to select an RMBE that best matches the characteristics on its TCP socket end point. The RMBE associated with a TCP socket endpoint must have a Receive buffer that is at least as large as the TCP receive buffer size in effect for that connection. The receive buffer size can be determined by what is specified explicitly by the application using `setsockopt()` or implicitly via the system configured default value. This will allow sufficient data to be RDMA written by the SMC-R peer to fill an entire receive buffer size worth of data on a given data flow. Given that each RMB must have fixed length RMBEs this implies that an SMC-R end point may need to maintain multiple RMBs of various sizes for SMC-R connections on a given SMC link and can then select an RMBE that most closely fits a connection.

##### 4.2. RMB and RMBE format

An RMB is a virtual memory buffer whose backing real memory is pinned, which is divided into a whole number of equal sized RMB Elements (RMBEs). Each RMBE begins with a four byte eye catcher for diagnostic and service purposes, followed by the receive data buffer. The contents of this diagnostic eyecatcher are implementation dependent and should be used by the local SMC-R peer to check for overlay errors by verifying an intact eyecatcher with every RMBE access.

The RMBE is a wrapping receive buffer for receiving RDMA writes from the peer. Cursors, as described below, are exchanged between peers to manage and track RDMA writes and local data reads from the RMBE for a TCP connection.

##### 4.3. RMBE control information

RMBE control information consists of consumer and producer cursors, wrap counts, CDC message sequence numbers, control flags such as urgent data and writer blocked indicators, and TCP connection

information such as termination flags. This information is exchanged between SMC-R peers using CDC messages, which are passed using RDMA message passing with inline data, with the control information contained in the inline data. A TCP/IP stack implementing SMC-R must receive and store this information in its internal data structures as it is used to manage the RMBE and its data buffer.

The format and contents of the CDC message is described in detail in 4.3. RMBE control information. The following is a high level description of what this control information contains.

- o Connection state flags such as sending done, connection closed, failover data validation, and abnormal close
- o A sequence number that is managed by the sender. This sequence number starts at 1, is increased each send, and wraps to 0. This sequence number tracks the CDC message sent and is not related to the number of bytes sent. It is used for failover data validation.
- o Producer cursor: a wrapping offset into the receiver's RMBE data area. Set by the peer that is writing into the RMBE, it points to where the writing peer will write the next byte of data into an RMBE. This cursor is accompanied by a wrap sequence number to help the RMBE owner (the receiver) identify full window size wrapping writes. Note that this cursor must account for (i.e., skip over) the RMBE eyecatcher that is in the beginning of the data area.
- o Consumer cursor: a wrapping offset into the receiver's RMBE data area. Set by the owner of the RMBE (the peer that is reading from it), this cursor points to the offset of the next byte of data to be consumed by the peer in its own RMBE. The sender cannot write beyond this cursor into the receiver's RMBE without causing data loss. Like the producer cursor, this is accompanied by a wrap count to help the writer identify full window size wrapping reads. Note that this cursor must account for (i.e., skip over) the RMBE eyecatcher that is in the beginning of the data area.
- o Data flags such as urgent data, writer blocked indicator, and cursor update requests.

#### 4.4. Use of RMBEs

##### 4.4.1. Initializing and accessing RMBEs

The RMBE eyecatcher is initialized by the RMB owner prior to assigning it to a specific TCP connection and communicating its RMB

index to the SMC-R partner. After an RMBE index is communicated to the SMC-R partner the RMBE can only be referenced in "read only mode" by the owner and all updates to it are performed by the remote SMC-R partner via RDMA write operations.

Initialization of an RMBE must include the following:

- o Zeroing out the entire RMBE receive buffer, which helps minimize data integrity issues (e.g. data from a previous connection somehow being presented to the current connection).
- o Setting the beginning RMBE eye catcher. This eye catcher plays an important role in helping detect accidental overlays of the RMBE. The RMB owner must always validate these eye catchers before each new reference to the RMBE. If the eye catchers are found to be corrupted the local host must reset the TCP connection associated with this RMBE and log the appropriate diagnostic information.

#### 4.4.2. RMB element reuse and conflict resolution

RMB elements can be reused once their associated TCP and SMC-R connections are terminated. Under normal and abnormal SMC-R connection termination processing both SMC-R peers must explicitly acknowledge that they are done using an RMBE before that element can be freed and reassigned to another SMC-R connection instance. For more details on SMC-R connection termination refer to section 4.8. However, there are some error scenarios where this 2 way explicit acknowledgement may not be completed. In these scenarios (mentioned explicitly elsewhere in this document) an RMBE owner may chose to re-assign this RMBE to a new SMC-R connection instance on this SMC link group. When this occurs the partner SMC-R peer must detect this condition during SMC-R rendezvous processing when presented with an RMBE that it believes is already in use for a different SMC-R connection. In this case, the SMC-R peer must abort the existing SMC-R connection associated with this RMBE. The abort processing Resets the TCP connection (if it is still active) but it must not attempt to perform any RDMA writes to this RMBE and must also ignore any data sitting in the local RMBE associated with the existing connection. It then proceeds to free up the local RMBE and notify the local application that the connection is being abnormally reset.

The remote SMC-R peer then proceeds to normal processing for this new SMC-R connection.

#### 4.5. SMC-R protocol considerations

The following sections describe considerations for the SMC-R protocol as compared to the TCP protocol.

##### 4.5.1. SMC-R protocol optimized window size updates

An SMC-R receiver host sends its Consumer Cursor information to the sender to convey the progress that the receiving application has made in consuming the sent data. The difference between the writer's Producer Cursor and the associated receiver's Consumer Cursor indicates the window size available for the sender to write into. This is somewhat similar to TCP window update processing and therefore has some similar considerations, such as silly window syndrome avoidance, whereby the TCP protocol has an optimization that minimizes the overhead of very small, unproductive window size updates associated with sub-optimal socket applications consuming very small amount of data on every receive() invocation. For SMC-R, the receiver only updates its Consumer Cursor via a unique CDC message under the following conditions:

- o The current window size (from a sender's perspective) is less than half of the Receive Buffer space and the Consumer Cursor update will result in a minimum increase in the window size of 10% of the Receive buffer space. Some examples:
  - a. Receive Buffer size: 64K, Current window size (from a sender's perspective): 50K. No need to update the Consumer Cursor. Plenty of space is available for the sender.
  - b. Receive Buffer size: 64K, Current window size (from a sender's perspective): 30K, Current window size from a receiver's perspective: 31K. No need to update the Consumer Cursor; even though the sender's window size < 1/2 of the 64K, the window update would only increase that by 1K which is < 1/10th of the 64K buffer size.
  - c. Receive Buffer size: 64K, Current window size (from a sender's perspective): 30K, Current window size from a receiver's perspective: 64K. The receiver updates the Consumer Cursor (sender's window size < 1/2 of the 64K, the window update would increase that by > 6.4K).

- o The receiver must always include a Consumer Cursor update whenever it sends a CDC message to the partner for another flow (i.e. send flow in the opposite direction). This allows the window size update to be delivered with no additional overhead. This is somewhat similar to TCP DelayAck processing and quite effective for request/response data patterns.
- o If a peer has set the B-bit in a CDC message then any consumption of data by the receiver causes a CDC message to be sent updating the consumer cursor until that a CDC message with that bit cleared is received from the peer.
- o The optimized window size updates are overridden when the sender sets the Consumer Cursor Update Requested flag in a CDC message to the receiver. When this indicator is on the consumer must send a Consumer Cursor update immediately when data is consumed by the local application or if the cursor has not been updated for a while (i.e. local copy consumer cursor does not match the last consumer cursor value sent to the the partner). This allows the sender to perform optional diagnostics for detecting a stalled receiver application (data has been sent but not consumed). It is recommended that the Consumer Cursor Update Requested flag only be sent for diagnostic procedures as it may result in non-optimal data path performance.

#### 4.5.2. Small data sends

The SMC-R protocol makes no special provisions for handling small data segments sent across a stream socket. Data is always sent if sufficient window space is available. There are no special provisions for coalescing small data segments, similar to the TCP Nagle algorithm.

An implementation of SMC-R may optimize its sending processing by coalescing outbound data for a given SMC-R connection so that it can reduce the number of RDMA write operations it performed in a similar fashion to Nagle's algorithm. However, any such coalescing would require a timer on the sending host that would ensure that data was eventually sent. And the sending host would have to opt out of this processing if Nagle's algorithm had been disabled (programmatically or via system configuration).

#### 4.5.3. TCP Keepalive processing

TCP keepalive processing allows applications to direct the local TCP/IP host to periodically "test" the viability of an idle TCP connection. Since SMC-R connections have both a TCP representation

along with an SMC-R representation there are unique keepalive processing considerations:

- o SMC-R layer keepalive processing: If keepalive is enabled for an SMC-R connection the local host maintains a keepalive timer that reflects how long an SMC-R connection has been idle. The local host also maintains a timestamp of last activity for each SMC link (for any SMC-R connection on that link). When it is determined that an SMC-R connection has been idle longer than the keepalive interval the host checks whether the SMC-R link has been idle for a duration longer than the keepalive timeout. If both conditions are met, the local host then performs a Test Link LLC command to test the viability of the SMC link over the RoCE fabric (RC-QPs). If a Test Link LLC command response is received within a reasonable amount of time then the link is considered viable and all connections using this link are considered viable as well. If however a response is not received in a reasonable amount of time or there's a failure in sending the Test Link LLC command then this is considered a failure in the SMC link and failover processing to an alternate SMC link must be triggered. If no alternate SMC link exists in the SMC link group then all the SMC-R connections on this link are abnormally terminated by resetting the TCP connections represented by these SMC-R connections. Given that multiple SMC-R connections can share the same SMC link, implementing an SMC link level probe using the Test Link LLC command will help reduce the amount of unproductive keepalive traffic for SMC-R connections; as long as some SMC-R connections on a given SMC link are active (i.e. have had I/O activity within the keepalive interval) then there is no need to perform additional link viability testing.
- o TCP layer keepalives processing: Traditional TCP "keepalive" packets are not as relevant for SMC-R connections given that the TCP path is not used for these connections once the SMC-R rendezvous processing is completed. All SMC-R connections by default have associated TCP connections that are idle. Are TCP keepalive probes still needed for these connections? There are two main scenarios to consider:
  1. TCP keepalives that are used determine whether the peer TCP endpoint is still active. This is not needed for SMC-R connections as the SMC-R level keepalives mentioned above will determine whether the remote endpoint connections are still active.

2. TCP keepalives that are used to ensure that TCP connections traversing an intermediate proxy maintain an active state. For example, stateful firewalls typically maintain state representing every valid TCP connection that traverses the firewall. These types of firewalls are known to expire idle connections by removing their state in the firewall to conserve memory. TCP keepalives are often used in this scenario to prevent firewalls from timing out otherwise idle connections. When using SMC-R, both end points must reside in the same layer 2 network (i.e. the same subnet). As a result, firewalls can not be injected in the path between two SMC-R endpoints. However, other intermediate proxies, such as TCP/IP layer load balancers may be injected in the path of two SMC-R endpoints. These types of load balancers also maintain connection state so that they can forward TCP connection traffic to the appropriate cluster end point. When using SMC-R these TCP connections will appear to be completely idle making them susceptible to potential timeouts at the LB proxy. As a result, for this scenario, TCP keepalives may still be relevant.

The following are the TCP level keepalive processing requirements for SMC-R enabled hosts:

- o SMC-R peers should allow TCP keepalives to flow on the TCP path of SMC-R connections based on existing TCP keepalive configuration and programming options. However, it is strongly recommended that platforms provide the ability to specify very granular keepalive timers (for example, single digit second timers) should consider providing a configuration option that limits the minimum keepalive timer that will be used for TCP layer keepalives on SMC-R connections. This is important to minimize the amount of TCP keepalive packets transmitted in the network for SMC-R connections.
- o SMC-R peers must always respond to inbound TCP layer keepalives (by sending ACKs for these packets) even if the connection is using SMC-R. Typically, once a TCP connection has completed the SMC-R rendezvous processing and using SMC-R for data flows, no new inbound TCP segments are expected on that TCP connection other than TCP termination segments (FIN, RST, etc). TCP keepalives are the one exception that must be supported. And since TCP keepalive probes do not carry any application layer data this has no adverse impact on the application's inbound data stream.

#### 4.6. TCP connection failover between SMC-R links

A peer may change which SMC-R link within a link group it sends its writes over in the event of a link failure. Since each peer independently chooses which link to send writes over for a specific TCP connection, this process is done independently by each peer.

##### 4.6.1. Validating data integrity

Even though RoCE is a reliable transport there is a small subset of failure modes that could cause unrecoverable loss of data. When an RNIC acknowledges receipt of an RDMA write to its peer, that creates a write completion event to the sending peer, which allows the sender to release any buffers it is holding for that write. In normal operation and in most failures, this operation is reliable.

However there are failure modes possible in which a receiving RNIC has acknowledged an RDMA write but then was not able to place the received data into its host memory, for example a sudden, disorderly failure of the interface between the RNIC and the host. While rare, these types of events must be guarded against to ensure data integrity. The process for switching SMC-R links during failover that is described in this section guards against this possibility, and is mandatory.

Each peer must track the current state of the CDC sequence numbers for a TCP connection. The sender must keep track of SS, which is the sequence number of the CDC message that described the last write acknowledged by the peer RNIC. In other words, SS describes the last write that the sender believes its peer has successfully received. The receiver must keep track of SR, the sequence number of the CDC message that described last write that it has successfully received, i.e., the data has been successfully placed into an RMBE.

When an RNIC fails and the sender changes SMC-R links, the sender must first send a CDC message with the 'F' flag set over the new SMC-R link. This is the failover data validation message. The sequence number in this CDC message is equal to SS. The CDC message key, the length, and SMC-R alert token are the only other fields in this CDC message that are significant. No reply is expected from this validation message, and once the sender has sent it, the sender may resume sending on the new SMC-R link as described in 4.6.2. below

Upon receipt of the failover validation message, the receiver must verify that its SR value for the TCP connection is equal to or greater than the sequence number in the failover validation message. If so, no further action is required and the TCP connection resumes

on the new SMC-R link. If SR is less than the sequence number value in the validation message, data has been lost and the receiver must immediately reset the TCP connection.

#### 4.6.2. Resuming the TCP connection on a new SMCR link

When a connection is moved to a new SMC-R link and the failover validation message has been sent, the sender can immediately resume normal transmission. In order to preserve the application message stream the sender must replay any RDMA writes (and their associated CDC messages) that were in progress or failed when the previous SMC-R link failed, before sending new data on the new SMC-R link. The sender has two options for accomplishing this:

- o Preserve the sequence numbers "as is": Retry all failed and pending operations as they were originally done, including reposting all associated RDMA write operations and their associated CDC messages without making any changes. Then resume sending new data using new sequence numbers.
- o Combine pending messages and possibly add new data: Combine failed and pending messages into a single new write with a new sequence number. This allows the sender to combine pending messages into fewer operations. As a further optimization this write can also include new data, as long as all failed and pending data is also included. If this approach is taken, the sequence number must be increased beyond the last failed or pending sequence number.

#### 4.7. RMB data flows

The following sections describe the RDMA wire flows for the SMC-R protocol after a TCP connection has switched into SMC-R mode (i.e. SMC-R rendezvous processing is complete and a pair of RMB elements has been assigned and communicated by the SMC-R peers). The ladder diagrams below include the following:

- o RMBE control information kept by each peer. Only a subset of the information is depicted, specifically only the fields that reflect the stream of data written by Host A and read by Host B.
- o Time line 0-x that shows the wire flows in a time relative fashion
- o Note that RMBE control information is only shown in a time interval if its value changed (otherwise assume the value is unchanged from previously depicted value)

- o The local copy of the producer and consumer cursors that is maintained by each host is not depicted in these figures. Note that the cursor values in the diagram reflect the necessity of skipping over the eyecatcher in the RMBE data area. They start and wrap at 4, not 0.

#### 4.7.1. Scenario 1: Send flow, window size unconstrained

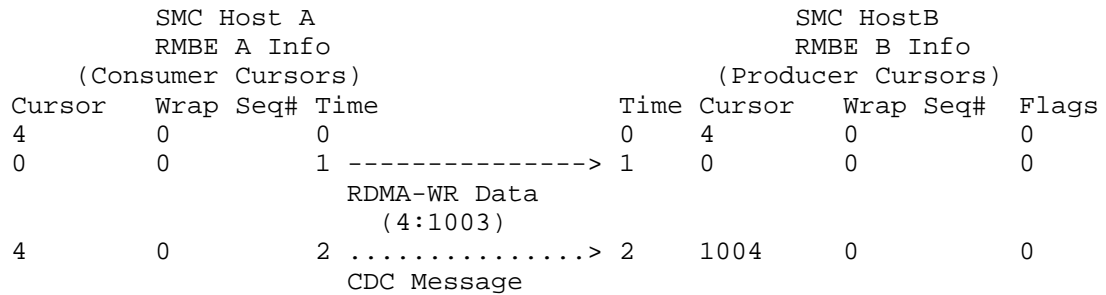


Figure 16 Scenario 1: Send flow, window size unconstrained

#### Scenario assumptions:

- o Kernel implementation
- o New SMC-R connection, no data has been sent on the connection
- o Host A: Application issues send for 1,000 bytes to Host B
- o Host B: RMBE receive buffer size is 10,000, application has issued a recv for 10,000 bytes

#### Flow description:

1. Application issues send() for 1,000 bytes, SMC-R layer copies data into a kernel send buffer. It then schedules an RDMA write operation to move the data into the peer's RMBE receive buffer, at relative position 4-1003 (to skip the four byte eyecatcher in the RMBE data area). Note that no immediate data or alert (i.e. interrupt) is provided to host B for this RDMA operation.

2. Host A sends a CDC message to update the Producer Cursor to byte 1004. This CDC message will deliver an interrupt to Host B. At this point, the SMC-R layer can return control back to the application. Host B, once notified of the completion of the previous RDMA operation, locates the RMBE associated with the RMBE alert token that was included in the message and proceeds to perform normal receive side processing, waking up the suspended application read thread, copying the data into the application's receive buffer, etc. It will use the Producer Cursor as an indicator of how much data is available to be delivered to the local application. After this processing is complete, the SMC-R layer will also update its local Consumer Cursor to match the Producer Cursor (i.e. indicating that all data has been consumed). Note that a message to the peer updating the Consumer Cursor is not needed at this time as the window size is unconstrained ( $> 1/2$  of the receive buffer size). The window size is calculated using by taking the difference between the Producer and the Consumer cursors in the RMBEs ( $10,000 - 1,004 = 8,996$ ).

#### 4.7.2. Scenario 2: Send/Receive flow, window unconstrained

SMC Host A				SMC HostB				
RMBE A Info				RMBE B Info				
(Consumer Cursors)				(Producer Cursors)				
Cursor	Wrap	Seq#	Time	Time	Cursor	Wrap	Seq#	Flags
4	0	0		0	4	0		0
0	0	1	----->	1	0	0		0
RDMA-WR Data (4:1003)								
4	0	2	.....>	2	1004	0		0
CDC Message								
0	0	3	<-----	3	1004	0		0
RDMA-WR Data (4:503)								
1004	0	4	<.....	4	1004	0		0
CDC Message								

Figure 17 Scenario 2: Send/Recv flow, window size unconstrained

##### Scenario assumptions:

- o New SMC-R connection, no data has been sent on the connection
- o Host A: Application issues send for 1,000 bytes to Host B
- o Host B: RMBE receive buffer size is 10,000, application has already issued a recv for 10,000 bytes. Once the receive is completed, the application sends a 500 byte response to Host A.

##### Flow description:

1. Application issues send() for 1,000 bytes, SMC-R layer copies data into a kernel send buffer. It then schedules an RDMA write operation to move the data into the peer's RMBE receive buffer, at relative position 4-1003. Note that no immediate data or alert (i.e. interrupt) is provided to host B for this RDMA operation.
2. Host A sends a CDC message to update the Producer Cursor to byte 1004. This CDC message will deliver an interrupt to Host B. At this point, the SMC-R layer can return control back to the application.

3. Host B, once notified of the receipt of the previous CDC message, locates the RMBE associated with the RMBE alert token and proceeds to perform normal receive side processing, waking up the suspended application read thread, copying the data into the application's receive buffer, etc. After this processing is complete, the SMC-R layer will also update its local Consumer Cursor to match the Producer Cursor (i.e. indicating that all data has been consumed). Note that an update of the Consumer Cursor to the peer is not needed at this time as the window size is unconstrained ( $> 1/2$  of the receive buffer size). The application then performs a send() for 500 bytes to Host A. The SMC-R layer will copy the data into a kernel buffer and then schedule an RDMA Write into the partner's RMBE receive buffer. Note that this RDMA write operation includes no immediate data or notification to Host A.
4. Host B sends a CDC message to update the partner's RMBE Control information with the latest Producer Cursor (set to 503 and not shown in the diagram above) and to also inform the peer that the Consumer Cursor value is now 1004. It also updates the local Current Consumer Cursor and Last Sent Consumer Cursor to 1004. This CDC message includes notification since we are updating our Producer Cursor which requires attention by the peer host.

#### 4.7.3. Scenario 3: Send Flow, window constrained

SMC Host A				SMC HostB				
RMBE A Info				RMBE B Info				
(Consumer Cursors)				(Producer Cursors)				
Cursor	Wrap	Seq#	Time	Time	Cursor	Wrap	Seq#	Flags
4	0	0		0	4	0		0
4	0	1	----->	1	4	0		0
RDMA-WR Data (4:3003)								
4	0	2	.....>	2	3004	0		0
CDC Message								
4	0	3		3	3004	0		0
4	0	4	----->	4	3004	0		0
RDMA-WR Data (3004:7003)								
4	0	5	.....>	5	7004	0		0
CDC Message								
7004	0	6	<.....	6	7004	0		0
CDC Message								

Figure 18 Scenario 3: Send Flow, window size constrained

Scenario assumptions:

- o New SMC-R connection, no data has been sent on this connection
- o Host A: Application issues send for 3,000 bytes to Host B and then another send for 4,000
- o Host B: RMBE receive buffer size is 10,000. Application has already issued a recv for 10,000 bytes

Flow description:

1. Application issues send() for 3,000 bytes, SMC-R layer copies data into a kernel send buffer. It then schedules an RDMA write operation to move the data into the peer's RMBE receive buffer, at relative position 4-3003. Note that no immediate data or alert (i.e. interrupt) is provided to host B for this RDMA operation.
2. Host A sends a CDC message to update its Producer Cursor to byte 3003. This CDC message will deliver an interrupt to Host B. At this point, the SMC-R layer can return control back to the application.
3. Host B, once notified of the receipt of the previous CDC message, locates the RMBE associated with the RMBE alert token and proceeds to perform normal receive side processing, waking up the suspended application read thread, copying the data into the application's receive buffer, etc. After this processing is complete, the SMC-R layer will also update its local Consumer Cursor to match the Producer Cursor (i.e. indicating that all data has been consumed). It will not however update the partner with this information as the window size is not constrained (10000-3000=7000 of available space). The application on Host B also issues a new recv() for 10,000.
4. On Host A, application issues a send() for 4,000 bytes. The SMC-R layer copies the data into a kernel buffer and schedules an async RDMA write into the peer's RMBE receive buffer at relative position 3003-7004. Note that no alert is provided to host B for this flow.
5. Host A sends a CDC message to update the Producer Cursor to byte 7004. This CDC message will deliver an interrupt to Host B. At this point, the SMC-R layer can return control back to the application.

6. Host B, once notified of the receipt of the previous CDC message, locates the RMBE associated with the RMBE alert token and proceeds to perform normal receive side processing, waking up the suspended application read thread, copying the data into the application's receive buffer, etc. After this processing is complete, the SMC-R layer will also update its local Consumer Cursor to match the Producer Cursor (i.e. indicating that all data has been consumed). It will then determine whether it needs to update the Consumer Cursor to the peer. The available window size is now 3,000 (10,000 - (Producer Cursor - Last Sent Consumer Cursor)) which is < 1/2 receive buffer size (10,000/2=5,000) and the advance of the window size is > 10% of the windows size (1,000). Therefore a CDC message is issued to update the Consumer Cursor to peer A.

#### 4.7.4. Scenario 4: Large send, flow control, full window size writes

SMC Host A				SMC HostB				
RMBE A Info				RMBE B Info				
(Consumer Cursors)				(Producer Cursors)				
Cursor	Wrap	Seq#	Time	Time	Cursor	Wrap	Seq#	Flags
1004	1	0		0	1004	1	0	
1004	1	1	----->	1	1004	1	0	
			RDMA-WR Data (1004:9999)					
1004	1	2	----->	2	1004	1	0	
			RDMA-WR Data (4:1003)					
1004	1	3	.....>	3	1004	2		Wrt
			CDC Message					
1004	2	4	<.....	4	1004	2		Wrt
			CDC Message					
1004	2	5	----->	5	1004	2		Wrt Blk
			RDMA-WR Data (1004:9999)					
1004	2	6	----->	6	1004	2		Wrt Blk
			RDMA-WR Data (4:1003)					
1004	2	7	.....>	7	1004	3		Wrt
			CDC Message					
1004	3	8	<.....	8	1004	3		Wrt
			CDC Message					

Figure 19 Scenario 4: Large send, flow control, full window size writes

Scenario assumptions:

- o Kernel implementation
- o Existing SMC-R connection, Host B's receive window size is fully open (Peer Consumer Cursor = Peer Producer Cursor).
- o Host A: Application issues send for 20,000 bytes to Host B
- o Host B: RMB receive buffer size is 10,000, application has issued a recv for 10,000 bytes

Flow description:

1. Application issues send() for 20,000 bytes, SMC-R layer copies data into a kernel send buffer (assumes send buffer space of 20,000 is available for this connection). It then schedules an RDMA write operation to move the data into the peer's RMBE receive buffer, at relative position 1004-9999. Note that no immediate data or alert (i.e. interrupt) is provided to host B for this RDMA operation.
2. Host A then schedules an RDMA write operation to fill the remaining 1000 bytes of available space in the peer's RMBE receive buffer, at relative position 4-1003. Note that no immediate data or alert (i.e. interrupt) is provided to host B for this RDMA operation. Also note that an implementation of SMC-R may optimize this processing by combining step 1 and 2 into a single RDMA Write operation (with 2 different data sources).
3. Host A sends CDC message to update the Producer Cursor to byte 1004. Since the entire receive buffer space is filled, the Producer Writer Blocked flag (WrtBlk indicator above) is set and the Producer Window Wrap Sequence Number (Producer WrapSeq# above) is incremented. This CDC message will deliver an interrupt to Host B. At this point, the SMC-R layer can return control back to the application.

4. Host B, once notified of the receipt of the previous CDC message, locates the RMBE associated with the RMBE alert token and proceeds to perform normal receive side processing, waking up the suspended application read thread, copying the data into the application's receive buffer, etc. In this scenario, Host B notices that the Producer Cursor has not been advanced (same value as Consumer Cursor), however, it notices that the Producer Window Wrap Size Sequence number is different from its local value (1) indicating that a full window of new data is available. All the data in the receive buffer can be processed, the first segment (1004-9999) followed by the second segment (4-1003). Because the Producer Writer Blocked indicator was set, Host B schedules a CDC message to update its latest information to the peer: Consumer Cursor (1004), Consumer Window Wrap Size Sequence Number (2: the current Producer Window Wrap Sequence Number is used).
5. Host A, upon receipt of the CDC message locates the TCP connection associated with the alert token, and upon examining the control information provided notices that Host B has consumed all of the data (based on the Consumer Cursor and the Consumer Window Wrap Size Sequence number) and initiates the next RDMA write to fill the receive buffer at offset 1003-9999.
6. Host A then moves the next 1000 bytes into the beginning of the receive buffer (4-1003) by scheduling an RDMA write operation. Note at this point there are still 8 bytes remaining to be written.
7. Host A then sends a CDC message to set the Producer Writer Blocked indicator and to increment the Producer Window Wrap Size Sequence Number (3).
8. Host B, upon notification completes the same processing as step 4 above, including sending a CDC message to update the peer to indicate that all data has been consumed. At this point Host A can write the final 8 bytes to host B's RMBE into positions 1004-1011 (not shown).

#### 4.7.5. Scenario 5: Send flow, urgent data, window size unconstrained

SMC Host A RMBE A Info (Consumer Cursors)				SMC HostB RMBE B Info (Producer Cursors)			
Cursor	Wrap	Seq#	Time	Time	Cursor	Wrap	Seq# Flag
1000	1	0		0	1000	1	0
1000	1	1	----->	1	1000	1	0
			RDMA-WR Data (1000:1499)				
1000	1	2	.....>	2	1500	1	UrgP
			CDC Message				UrgA
1500	1	3	<.....	3	1500	1	UrgP
			CDC Message				UrgA
1500	1	4	----->	4	1500	1	UrgP
			RDMA-WR Data (1500:2499)				UrgA
1500	1	5	.....>	5	2500	1	0
			CDC Message				

Figure 20 Scenario 5: send Flow, urgent data, window size open

Scenario assumptions:

- o Kernel implementation
- o Existing SMC-R connection, window size open, all data has been consumed by receiver.
- o Host A: Application issues send for 500 bytes with urgent data indicator (OOB) to Host B, then sends 1000 of normal data
- o Host B: RMBE Receive buffer size is 10,000, application has issued a recv for 10,000 bytes and is also monitoring the socket for urgent data

Flow description:

1. Application issues send() for 500 bytes of urgent data. SMC-R layer copies data into a kernel send buffer. It then schedules an RDMA write operation to move the data into the peer's RMBE receive buffer, at relative position 1000-1499. Note that no immediate data or alert (i.e. interrupt) is provided to host B for this RDMA operation.

2. Host A sends a CDC message to update its Producer Cursor to byte 1500 and to turn on the Producer Urgent Data Pending (UrgP) and Urgent Data Present (UrgA) flags. This CDC message will deliver an interrupt to Host B. At this point, the SMC-R layer can return control back to the application.
3. Host B, once notified of the receipt of the previous CDC message, locates the RMBE associated with the RMBE alert token, notices that the Urgent Data Pending flag is on and proceeds with Out of Band socket API notification. For example, satisfying any outstanding select() or poll() requests on the socket by indicating that urgent data is pending (i.e. by setting the exception bit on). The Urgent Data Present indicator allows Host B to also determine the position of the urgent data (Producer cursor points one byte beyond the last byte of urgent data). Host B can then perform normal receive side processing (including specific urgent data processing), copying the data into the application's receive buffer, etc. Host B then sends a CDC message to update the partner's RMBE Control area with its latest Consumer Cursor (1500). Note this CDC message must occur regardless of the current local window size that is available. The partner host (Host A) cannot initiate any additional RDMA writes until acknowledgement that the urgent data has been processed (or at least processed/remembered at the SMC-R layer).
4. Upon receipt of the message, Host A wakes up, sees that peer consumed all data up to and including the last byte of Urgent data and now resumes sending any pending data. In this case, the application had previously issued a send for 1000 bytes of normal data which would have been copied in the send buffer and control would have been returned to the application. Host A now initiates a RDMA write to move that data to the Peer's receive buffer at position 1500-2499.
5. Host A then sends a CDC message with inline data update its Producer Cursor value (2500) and turn off the Urgent Data Pending and Urgent Data Present flags. Host B wakes up, processes the new data (resumes application, copies data into the application receive buffer) and then proceeds to update the Local current consumer cursor (2500). Given that the window size is unconstrained there is no need for Consumer Cursor update in the peer's RMBE.

## 4.7.6. Scenario 6: Send flow, urgent data, window size closed

SMC Host A RMBE A Info (Consumer Cursors)				SMC HostB RMBE B Info (Producer Cursors)			
Cursor	Wrap	Seq#	Time	Time	Cursor	Wrap	Seq# Flag
1000	1	0		0	1000	2	Wrt Blk
1000	1	1	.....>	1	1000	2	Wrt Blk UrgP
			CDC Message				
1000	2	2	<.....	2	1000	2	Wrt Blk UrgP
			CDC Message				
1000	2	3	----->	3	1000	2	Wrt Blk UrgP
			RDMA-WR data 1 (1000:1499)				
1000	2	4	.....>	4	1500	2	UrgP UrgA
			CDC Message				
1500	2	5	<.....	5	1500	2	UrgP UrgA
			CDC Message				
1500	2	6	----->	6	1500	2	UrgP UrgA
			RDMA-WR data 1 (1500:2499)				
1000	2	7	.....>	7	2500	2	0
			CDC Message				

Figure 21 Scenario 6: Send flow, urgent data, window size closed

Scenario assumptions:

- o Kernel implementation
- o Existing SMC-R connection, window size closed, writer is blocked.
- o Host A: Application issues send for 500 bytes with urgent data indicator (OOB) to Host B, then sends 1000 of normal data.
- o Host B: RMBE Receive buffer size is 10,000, application has no outstanding recv() (for normal data) and is monitoring the socket for urgent data.

Flow description:

1. Application issues send() for 500 bytes of urgent data. SMC-R layer copies data into a kernel send buffer (if available). Since the writer is blocked (window size closed) it cannot send the data immediately. It then sends a CDC message to notify the peer of the Urgent Data Pending (UrgP) indicator (the Writer Blocked indicator remains on as well). This serves as a signal to Host B that urgent data is pending in the stream. Control is also returned to the application at this point.
2. Host B, once notified of the receipt of the previous CDC message, locates the RMBE associated with the RMBE alert token, notices that the Urgent Data Pending flag is on and proceeds with Out of Band socket API notification. For example, satisfying any outstanding select() or poll() requests on the socket by indicating that urgent data is pending (i.e. by setting the exception bit on). At this point it is expected that the application will enter urgent data mode processing, expeditiously processing all normal data (by issuing recv API calls) so that it can get to the urgent data byte. Whether the application has this urgent mode processing or not, at some point the application will consume some or all of the pending data in the receive buffer. When this occurs, Host B will also send a CDC message with inline data to update its Consumer Cursor and Consumer Window Wrap Sequence Number to the peer. In the example above, a full window worth of data was consumed.
3. Host A, once awakened by the message will notice that the window size is now open on this connection (based on the Consumer Cursor and the Consumer Window Wrap Sequence Number which now matches the Producer Window Wrap Sequence Number) and resume sending of the urgent data segment by scheduling an RDMA write into relative position 1000-1499.
4. Host A then sends a CDC message to advance its Producer Cursor (1500) and to also notify Host B of the Urgent Data Present (UrgA) indicator (and turn off the Writer Blocked indicator). This signals to Host B that the urgent data is now in the local receive buffer and that the Producer Cursor points to the last byte of urgent data.
5. Host B wakes up, processes the urgent data and once the urgent data is consumed sends a CDC message with inline data to update its Consumer Cursor (1500).

6. Host A wakes up, sees that Host B has consumed the sequence number associated with the urgent data and then initiates the next RDMA write operation to move the 1000 bytes associated with the next send() of normal data into the peer's receive buffer at position (1500-2499). Note that send() API would have likely completed earlier in the process by copying the 1000 bytes into a send buffer and returning back to the application even though we could not send any new data until the urgent data was processed and acknowledged by Host B.
7. Host A sends a CDC message to advance its Producer Cursor to 2500 and to reset the Urgent Data Pending and Present flags. Host B wakes up and processes the inbound data.

#### 4.8. Connection termination

Just as SMC-R connections are established using a combination of TCP connection establishment flows and SMC-R protocol flows, the termination of SMC-R connections also uses a similar combination of SMC-R protocol termination flows and normal TCP protocol connection termination flows. The following sections describe the SMC-R protocol normal and abnormal connection termination flows.

##### 4.8.1. Normal SMC-R connection termination flows

Normal SMC-R connection flows are triggered via the normal stream socket API semantics, namely by the application issuing a close() or shutdown() API. Most applications, after consuming all incoming data and after sending any outbound data will then issue a close() API to indicate that they are done both sending and receiving data. Some applications, typically a small percentage, make use of the shutdown() API that allows then to indicate that the application is done sending data, receiving data or both sending and receiving data. The main use of this API is scenarios where a TCP application wants to alert its partner end point that it is done sending data, yet is still receiving data on its socket (shutdown for Write). Issuing shutdown for both sending and receiving data is really no different than issuing a close() and can therefore be treated in a similar fashion. Shutdown for read is typically not a very useful operation and in normal circumstances does not trigger any network flows to notify the partner TCP end point of this operation.

These same trigger points will be used by the SMC-R layer to initiate SMC-R connections termination flows. The main design point for SMC-R normal connection flows is to use the SMC-R protocol to first shutdown the SMC-R connection and free up any SMC-R RDMA resources and then allow the normal TCP connection termination protocol (i.e.

FIN processing) to drive cleanup of the TCP connection. This design point is very important in ensuring that RDMA resources such as the RMBEs are only freed and reused when both SMC-R end points are completely done with their RDMA Write operations to the partner's RMBE.

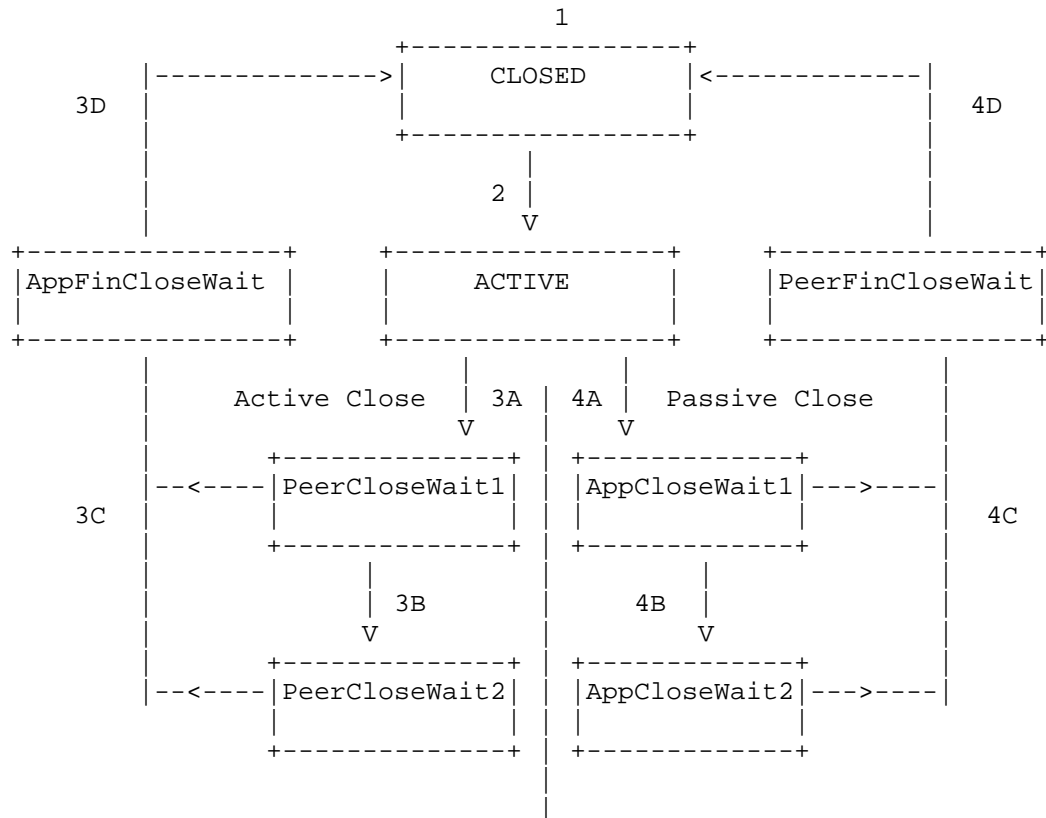


Figure 22 SMC-R connection states

Figure 23 describes the states that an SMC-R connection typically goes through. Note that there are variations to these states that can occur when an SMC-R connection is abnormally terminated, similar in a way to when a TCP connection is reset. The following are the high level state transitions for an SMC-R connection:

1. An SMC-R connection begins in the Closed state. This state is meant to reflect an RMBE that is not currently in use (was previously in use but no longer is or one that was never allocated)
2. An SMC-R connection progresses to the Active state once the SMC-R rendezvous processing has successfully completed, RMB element indices have been exchanged and SMC-R links have been activated. In this state, TCP connection is fully established, rendezvous processing has been completed and SMC-R peers can begin exchange of data via RDMA.
3. Active close processing (on SMC-R peer that is initiating the connection termination)

A. When an application on one of the SMC-R connection peers issues a close() or shutdown(write or both) the SMC-R layer on that host will initiate SMC-R connection termination processing. First if close() or shutdown(both) is issued it will check to see that there's no data in the local RMB element that has not been read by the application. If unread data is detected, the SMC-R connection must be abnormally reset - for more detail on this refer to "SMC-R connection reset". If no unread data is pending, it then checks to see whether any outstanding data is waiting to be written to the peer or if any outstanding RDMA writes for this SMC-R connection have not yet completed. If either of these two scenarios are true, an indicator that this connection is in a pending close state is saved in internal data structures representing this SMC-R connection and control is returned to the application. If all data to be written to the partner has completed this peer will send a CDC message to notify the peer of either the PeerConnectionClosed indicator (close or shutdown for both was issued) or the PeerDoneWriting indicator. This will provide stimulus to the partner SMC-R peer that the connection is terminating. At this point the local side of the SMC-R connection transitions in the PeerCloseWait1 state and control can be returned to the application. If this process could not be completed synchronously (close pending condition mentioned above) it is completed when all RDMA writes for data and control cursors have been completed.

B. At some point the SMC-R peer application (passive close) will consume all incoming data, realize that that partner is done sending data on this connection and proceed to initiate its own close of the connection once it has completed sending all data from its end. The partner application can initiate this connection termination processing via a close() or shutdown()

APIs. If the application does so by issuing a shutdown() for write, then the partner SMC-R layer will send a CDC message to notify the peer (active close side) of the PeerDoneWriting indicator. When the "active close" SMC-R peer wakes up as a result of the previous CDC message, it will notice that the PeerDoneWriting indicator is now on and transition to the PeerCloseWait2 state. This state indicates that the peer is done sending data and may still be reading data. The "active close" peer will also at this point need to ensure that any outstanding recv() calls for this socket are woken up and remember that that no more data is forthcoming on this connection (in case the local connection was shutdown() for write only)

C. This flow is a common transition from 3a or 3b above. When the SMC-R peer (passive close) consumes all data, updates all necessary cursors to the peer and the application closes its socket (close or shutdown for both) it will send a CDC message to the peer (the active close side) with the PeerConnectionClosed indicator set. At this point the connection can transition back to Closed state if the local application has already closed (or issued shutdown for both) the socket. Once in the Closed state, the RMBE can now be safely be reused for a new SMC-R connection. When the PeerConnectionClosed indicator is turned on, the SMC-R peer is indicating that it is done updating the partner's RMBE.

D. Conditional State: If the local application has not yet issued a close() or shutdown(both) yet, we need to wait until the application does so (ApplFinWaitState). Once it does, the local host will send a CDC message to notify the peer of the PeerConnectionClosed indicator and then transition to the Closed state.

4. Passive close processing (on SMC-R peer that receives an indication that the partner is closing the connection)

A. Upon receipt of an inbound RDMA write notice the SMC-R layer will detect that the PeerConnectionClosed indicator or PeerDoneWriting indicator is on. If any outstanding recv() calls are pending they are completed with an indicator that the partner has closed the connection (zero length data presented to application). If any pending data to be written and PeerConnectionClosed is on then an SMC-R connection reset must be performed. The connection then enters the ApplCloseWait1 state on the passive close side waiting for the local application to initiate its own close processing

B. If the local application issues a shutdown() for writing then the SMC-R layer will send a CDC message to notify the partner of the PeerDoneWriting indicator transition the local side of the SMC-R connection to the ApplCloseWait2 state.

C. When the application issues a close() or shutdown() for both, the local SMC-R peer will send a message informing the peer of the PeerConnectionClosed indicator and transition to the Closed state if the remote peer has also sent the local peer the PeerConnectionClosed indicator. If the peer has not sent the PeerConnectionClosed indicator, we transition into the PeerFinalCloseWait state.

D. The local SMC-R connection stays in this state until the peer sends the PeerConnectionClosed indicator in our RMBE. When the indicator is sent we transition to the Closed state and are then free to reuse this RMBE.

Note that each SMC-R peer needs to provide some logic that will prevent being stranded in termination state indefinitely. For example, if an Active Close SMC-R peer is in a PeerCloseWait (1 or 2) state awaiting the remote SMC-R peer to update its connection termination status it needs to provide a timer that will prevent it from waiting in that state indefinitely should the remote SMC-R peer not respond to this termination request. This could occur in error scenarios; for example, if the remote SMC-R peer suffered a failure prior to being able to respond to the termination request or the remote application is not responding to this connection termination request by closing its own socket. This latter scenario is similar to the TCP FINWAIT2 state that has been known to sometimes cause issues when remote TCP/IP hosts lose track of established connections and neglect to close them. Even though the TCP standards do not mandate a time out from the TCP FINWAIT2 state, most TCP/IP implementations implement a timeout for this state. A similar timeout will be required for SMC-R connections. When this timeout occurs, the local SMC-R peer performs TCP reset processing for this connection. However, no additional RDMA writes to the partner RMBE can occur at this point (we have already indicated that we are done updating the peer's RMBE). After the TCP connection is Reset the RMBE can be returned to the free pool for reallocation. See section 3.2.5 for more details.

Also note that it is possible to have two SMC-R end points initiate an Active close concurrently. In that scenario the flows above still apply, however, both end points follow the active close path (path 3).

#### 4.8.1.1. Abnormal SMC-R connection termination flows

Abnormal SMC-R connection termination can occur for a variety of reasons, including:

- o The TCP connection associated with an SMC-R connection is reset. In the TCP protocol either end point can send a RST segment to abort an existing TCP connection when error conditions are detected for the connection or the application overtly requests that the connection be reset.
- o Normal SMC-R connection termination processing has unexpectedly stalled for a given connection. When the stall is detected (connection termination timeout condition) an abnormal SMC-R connection termination flow is initiated.

In these scenarios it is very important that resources associated with the affected SMC-R connections are properly cleaned up to ensure that there are no orphaned resources and that resources can reliably be reused for new SMC-R connections. Given that SMC-R relies heavily on the RDMA Write processing, special care needs to be taken to ensure that an RMBE is no longer being used by a SMC-R peer before logically reassigning that RMBE to a new SMC-R connection.

When an SMC-R peer initiates a TCP connection reset it also initiates an SMC-R abnormal connection flow at the same time. The SMC-R peers explicitly signal their intent to abnormally terminate an SMC-R connection and await explicit acknowledgement that the peer has received this notification and has also completed abnormal connection termination on its end. Note that TCP connection reset processing can occur in parallel to these flows.

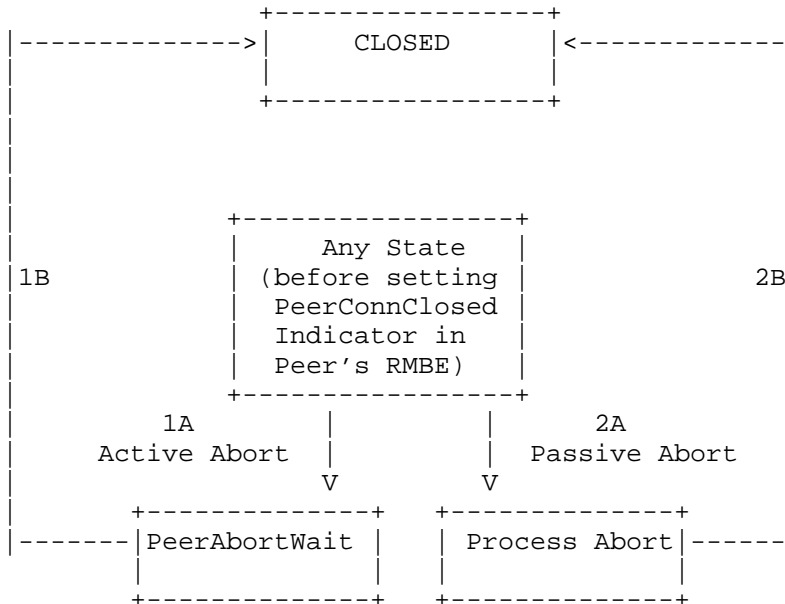


Figure 23 SMC-R abnormal connection termination state diagram

Figure 24 above shows the SMC-R abnormal connection termination state diagram:

1. Active abort designates the SMC-R peer that is initiating the TCP RST processing. At the time that the TCP RST is sent the active abort side must also
  - A. Send the PeerConnAbort indicator to the partner via RDMA messaging with inline data and then transition to the PeerAbortWait state. During this state it will monitor this SMC-R connection waiting for the peer to send its corresponding PeerConnAbort indicator but will ignore any other activity in this connection (i.e. new incoming data). It will also surface an appropriate error to any socket API calls issued against this socket (e.g. ECONNABORTED, ECONNRESET, etc.)
  - B. Once the peer sends the PeerConnAbort indicator to the local host, the local host can transition this SMC-R connection to the Closed state and reuse this RMBE. Note that the SMC-R peer that goes into the Active abort state must provide some protection

against staying in that state indefinitely should the remote SMC-R peer not respond by sending its own PeerConnAbort indicator to the local host. While this should be a rare scenario it could occur if the remote SMC-R peer (passive abort) suffered a failure right after the local SMC-R peer (active abort) sent the PeerConnAbort indicator. To protect against these types of failures, a timer can be set after entering the PeerAbortWait state and when if that timer pops before the peer has sent its local PeerConnAbort indicator (to the active abort side) then this RMBE can be returned to the free pool for possible re-allocation. See section See section 3.2.5 for more details.

2. Passive abort designates the SMC-R peer that is the recipient of an SMC-R abort from the peer designated by the PeerConnAbort indicator being sent by the peer in a CDC message. Upon receiving this request, the local peer must

- A. Indicate to the socket application that this connection has been aborted using the appropriate error codes, purge all in-flight data for this connection that is waiting to be read or waiting to be sent.

- B. Send a CDC message to notify the peer of the PeerConnAbort indicator and once that is completed transition this RMBE to the Closed state.

If an SMC-R peer receives a TCP RST for a given SMC-R connection it also initiates SMC-R abnormal connection termination processing if it has not already been notified (via the PeerConnAbort indicator) that the partner is severing the connection. It is possible to have two SMC-R endpoints concurrently be in an Active abort role for a given connection. In that scenario the flows above still apply but both end points take the active abort path (path 1).

#### 4.8.1.2. Other SMC-R connection termination conditions

The following are additional conditions that have implications of SMC-R connection termination:

- o A SMC-R peer being gracefully shut down. If an SMC-R peer supports a graceful shutdown operation it should attempt to terminate all SMC-R connections as part of shutdown processing. This could be accomplished via LLC Delete Link requests on all active SMC Links.

- o Abnormal termination of an SMC-R peer. In this example, there may be no opportunity for the host to perform any SMC-R cleanup processing. In this scenario it is up to the remote peer to detect a RoCE communications failure with the failing host. This could trigger an SMC link switch but that would also surface RoCE errors causing the remote host to eventually terminate all existing SMC-R connections to this peer.
- o Loss of RoCE connectivity between two SMC-R peers. If two peers are no longer reachable across any links in their SMC Link group then both peers perform a TCP reset for the connections, surface an error to the local applications and free up all QP resources associated with the link group.

## 5. Security considerations

### 5.1. VLAN considerations

The concepts and access control of virtual LANs (VLANs) must be extended to also cover the RoCE network traffic flowing across the ethernet.

The RoCE VLAN configuration and accesses must mirror the IP VLAN configuration and accesses over the CEE fabric. This means that hosts, routers and switches that have access to specific VLANs on the IP fabric must also have the same VLAN access across the RoCE fabric. In other words, the SMC-R connectivity will follow the same virtual network access permissions as normal TCP/IP traffic.

### 5.2. Firewall considerations

As mentioned above, the RoCE fabric inherits the same VLAN topology/access as the IP fabric. RoCE is a layer 2 protocol that requires both end points to reside in the same layer 2 network (i.e. VLAN). RoCE traffic can not traverse multiple VLANs as there is no support for routing RoCE traffic beyond a single VLAN. As a result, SMC-R communications will also be confined to peers that are members of the same VLAN. IP based firewalls are typically inserted between VLANs (or physical lans) and rely on normal IP routing to insert themselves in the data path. Since RoCE (and by extension SMC-R) is not routable beyond the local VLAN, there is no ability to insert a firewall in the network path of two SMC-R peers.

### 5.3. Host-based IP Filters

Because SMC-R maintains the TCP three-way handshake for connection setup before switching to RoCE out of band, existing IP filters that control connection setup flows remain effective in an SMC-R environment. IP filters that operate on traffic flowing in an active TCP connection are not supported, because the connection data does not flow over IP.

### 5.4. Intrusion Detection Services

Similar to IP filters, intrusion detection services that operate on TCP connection setups are compatible with SMC-R with no changes required. However once the TCP connection has switched to RoCE out of band, packets are not available for examination.

### 5.5. IP Security (IPSec)

IP Security is not compatible with SMC-R because there are no IP packets to operate on. TCP connections that require IP security must opt out of SMC-R.

### 5.6. TLS/SSL

TLS/SSL is preserved in an SMC-R environment. The TLS/SSL layer resides above the SMC-R layer and outgoing connection data is encrypted before being passed down to the SMC-R layer for RDMA write. Similarly, incoming connection data goes through the SMC-R layer encrypted and is decrypted by the TLS/SSL layer as it is today.

The TLS/SSL handshake messages flow over the TCP connection after the connection has switched to SMC-R, so are exchanged using RDMA writes by the SMC-R layer, transparently to the TLS/SSL layer.

## 6. IANA considerations

The scarcity of TCP option codes available for assignment is understood and this architecture uses experimental TCP options following the conventions of RFC 6994 "Shared Use of Experimental TCP Options".

If this protocol achieves wide acceptance a discrete option code may be requested by subsequent versions of this protocol.

## 7. References

### 7.1. Normative References

- [ROCE] RDMA over Converged Ethernet specification, URL,  
[http://members.infinibandta.org/kwspub/spec/Annex\\_RoCE\\_final.pdf](http://members.infinibandta.org/kwspub/spec/Annex_RoCE_final.pdf)
- [IBTA] Infiniband Architecture specification, URL,  
<http://www.infinibandta.org/specs>
- [RFC793] University of Southern California Information Services  
Institute, "Transmission Control Protocol", RFC 793,  
September 1981.
- [RFC4727] Fenner B., "Experimental Values in IPv4, IPv6, ICMPv4,  
ICMPv6, UDP, and TCP Headers", RFC 4727, November 2006.

### 7.2. Informative References

- [RFC 6994] Touch, J., "Shared use of Experimental TCP Options",  
draft URL, <https://tools.ietf.org/html/rfc6994>

## 8. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

## 9. Conventions used in this document

In the rendezvous flow diagrams, dashed lines (----) are used to indicate flows over the TCP/IP fabric and dotted lines (....) are used to indicate flows over the RoCE fabric.

In the data transfer ladder diagrams, dashed lines (----) are used to indicate RDMA write operations and dotted lines (....) are used to indicate CDC messages, which are RDMA messages with inline data that contain control information for the connection.

## Appendix A. Formats

### A.1. TCP option

The SMC-R TCP option is formatted in accordance with RFC 6994 "Shared Use of Experimental TCP Options". The ExID value is IBM-1047 (EBCDIC) encoding for 'SMCR'

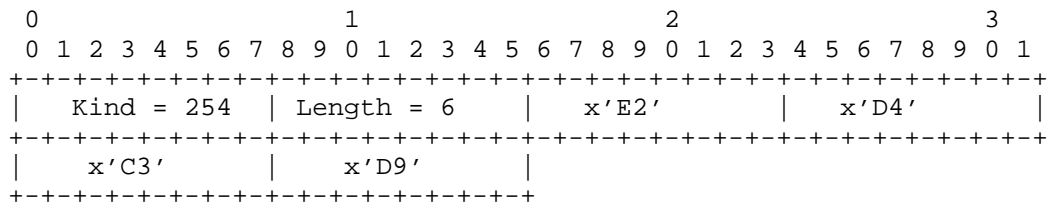


Figure 24 SMC-R TCP option format

### A.2. CLC messages

The following rules apply to all CLC messages:

General rules on formats:

- o Reserved fields must be set to zero and not validated
- o Each message has an eyecatcher at the start and another eyecatcher at the end. These must both be validated by the receiver.
- o SMC version indicator: The only SMC-R version defined in this architecture is version 1. In the future, if peers have a mismatch of versions, the lowest common version number is used.

#### A.2.1. Peer ID format

All CLC messages contain a peer ID that uniquely identifies an instance of a TCP/IP stack. This peer ID is required to be universally unique across TCP/IP stacks and instances (including restarts) of TCP/IP stacks.

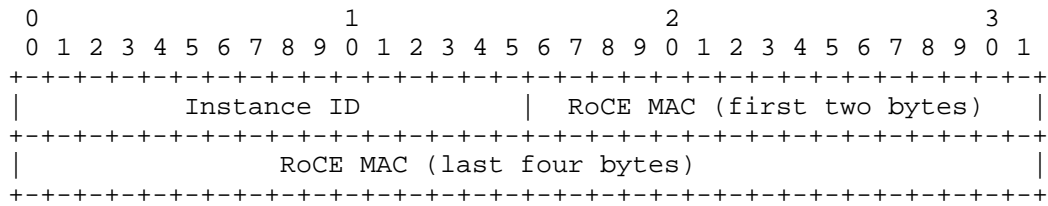


Figure 25 Peer ID format

#### Instance ID

A two-byte instance count that ensures that if the same RNIC MAC is later used in the peer ID for a different TCP/IP stack, for example if an RNIC is redeployed to another stack, the values are unique. It also ensures that if a TCP/IP stack is restarted, the instance ID changes. Value is implementation defined, with one suggestion being two bytes of the system clock.

#### RoCE MAC

The RoCE MAC address for one of the peer's RNICs. Note that in a virtualized environment this will be the virtual MAC of one of the peer's RNICs.

## A.2.2. SMC Proposal CLC message format

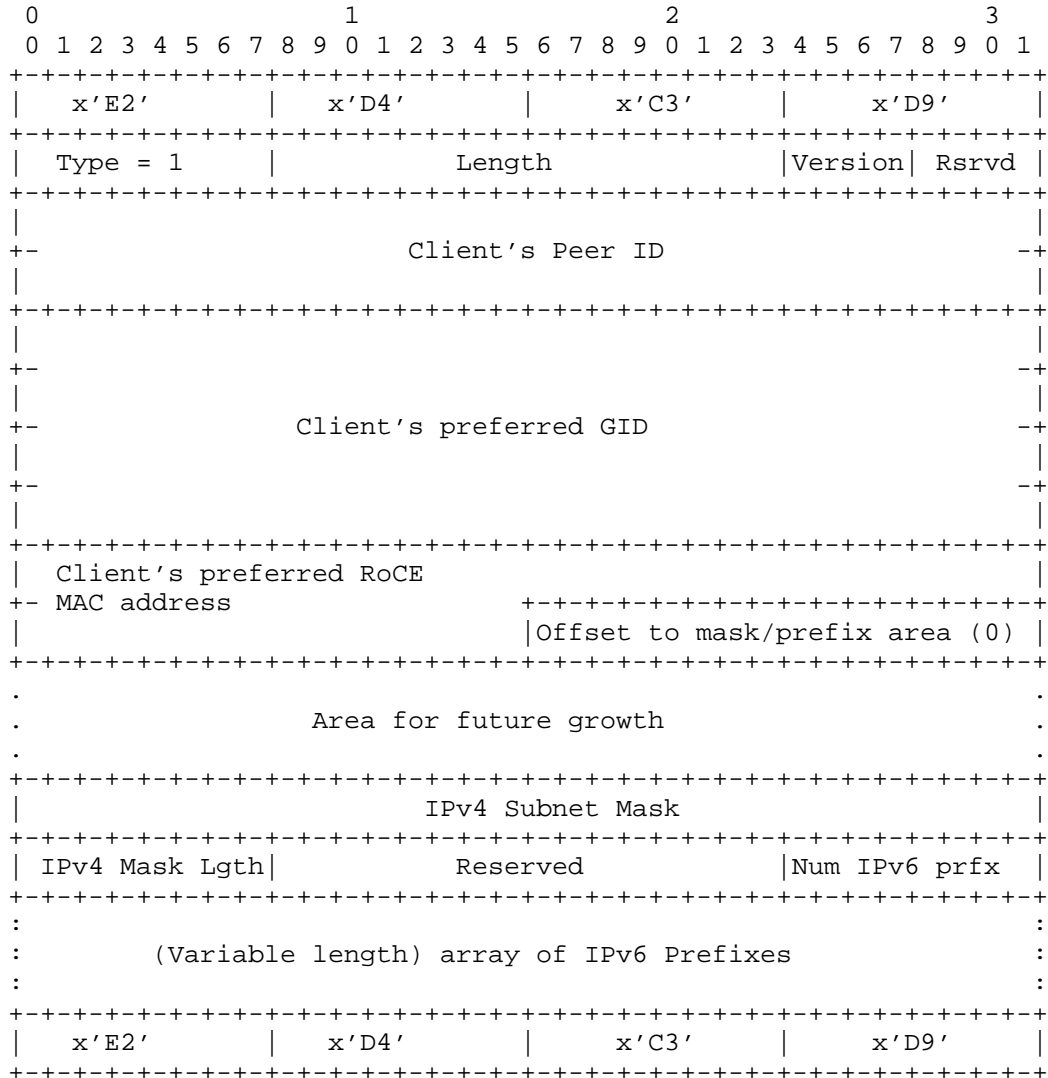


Figure 26 SMC Proposal CLC message format

The fields present in the SMC Proposal CLC message are:

Eyecatchers

Like all CLC messages, the SMC Proposal has beginning and ending eyecatchers to aid with verification and parsing. The hex digits spell 'SMCR' in IBM-1047 (EBCDIC)

Type

CLC message type 1 indicates SMC Proposal

Length

The length of this CLC message. If this an IPv4 flow, this value is 52. Otherwise it is variable depending upon how many prefixes are listed.

Version

Version of the SMC-R protocol. Version 1 is the only currently defined value

Client's Peer ID

As described in A.2.1. above

Client's preferred RoCE GID

This is the IPv6 address of the client's preferred RNIC on the RoCE fabric

Client's preferred RoCE MAC address

The MAC address of the client's preferred RNIC on the RoCE fabric. It is required as some operating systems do not have neighbor discovery or ARP support for RoCE RNICs.

Offset to mask/prefix area

Provides the number of bytes that must be skipped after this field, to access the IPv4 Subnet Mask and the fields that follow it. Allows for future growth of this signal. In this version of the architecture, this value is always zero.

Area for future growth

In this version of the architecture, this field does not exist. This indicates where additional information may be inserted into the signal in the future. "The Offset to mask/prefix area" field must be used to skip over this area.

## IPv4 Subnet mask

If this message is flowing over an IPv4 TCP connection, the value of the subnet mask associated with the interface the client sent this message over. If this an IPv6 flow this field is all zeroes.

This field, along with all fields that follow it in this signal, must be accessed by skipping the number of bytes listed in the "Offset to mask/prefix area" field after the end of that field.

## IPv4 Mask Lgth

If this message is flowing over an IPv4 TCP connection, the number of significant bits in the IPv4 subnet mask. If this an IPv6 flow, this field is zero.

## Num IPv6 prfx

If this message is flowing over an IPv6 TCP connection, the number of IPv6 prefixes that follow, with a maximum value of 8. if this is an IPv4 flow this field is zero and is immediately followed by the ending eyecatcher.

## Array of IPv6 Prefixes

For IPv6 TCP connections, a list of the IPv6 prefixes associated with the network the client sent this message over, up to a maximum of 8 prefixes.

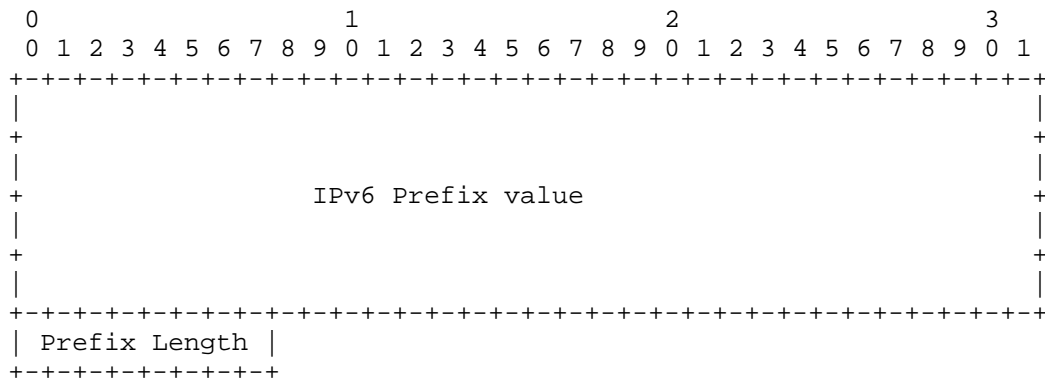


Figure 27 Format for IPv6 Prefix array element

## A.2.3. SMC Accept CLC message format

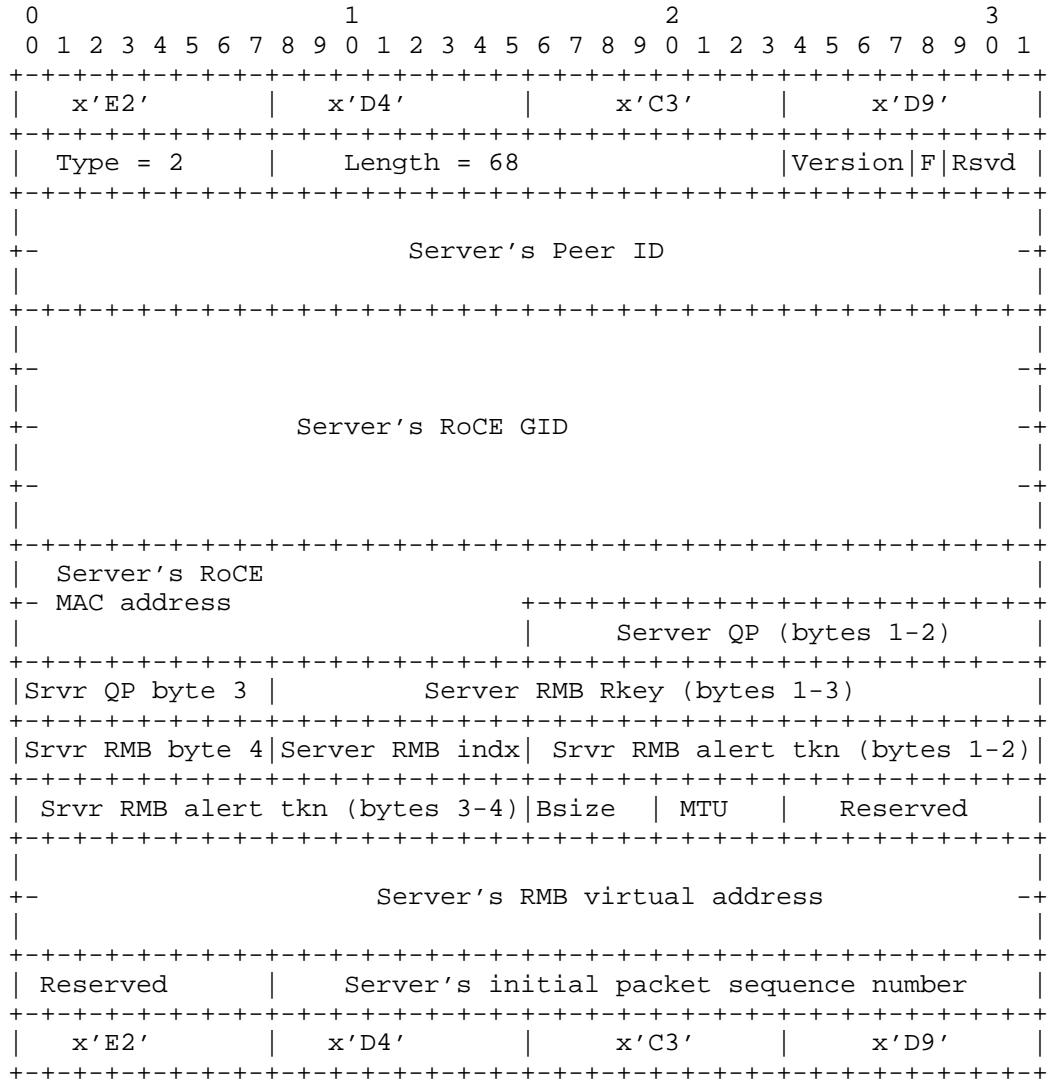


Figure 28 SMC Accept CLC message format

The fields present on the SMC Accept CLC message are:

Eyecatchers

Like all CLC messages, the SMC Accept has beginning and ending eyecatchers to aid with verification and parsing. The hex digits spell 'SMCR' in IBM-1047 (EBCDIC)

Type

CLC message type 2 indicates SMC Accept

Length

The SMC Accept CLC message is 68 bytes long

Version

Version of the SMC-R protocol. Version 1 is the only currently defined value.

F-bit

First Contact flag: A 1-bit flag that indicates that the server believes this TCP connection is the first SMC-R contact for this link group

Server's Peer ID

As described in A.2.1. above

Server's RoCE GID

This is the IPv6 address of the RNIC that the server chose for this SMC Link

Server's RoCE MAC address

The MAC address of the server's RNIC for the SMC link. It is required as some operating systems do not have neighbor discovery or ARP support for RoCE RNICs.

Server's QP number

The number for the reliably connected queue pair that the server created for this SMC link

Server's RMB Rkey

The RDMA Rkey for the RMB that the server created or chose for this TCP connection

#### Server's RMB element index

This indexes which element within the server's RMB will represent this TCP connection

#### Server's RMB element alert token

A platform defined, architecturally opaque token that identifies this TCP connection. Added by the client as immediate data on RDMA writes from the client to the server to inform the server that there is data for this connection to retrieve from the RMB element

#### Bsize:

Server's RMB element buffer size in four bits compressed notation:  $x=4$  bits. Actual buffer size value is  $(2^{(x+4)}) * 1K$ . Smallest possible value is 16K. Largest size supported by this architecture is 512K.

#### MTU

An enumerated value indicating this peer's QP MTU size. The two peers exchange this value and the minimum of the peer's value will be used for the QP. This field should only be validated on a first contact exchange.

The enumerated MTU values are:

- 0: reserved
- 1: 256
- 2: 512
- 3: 1024
- 4: 2048
- 5: 4096
- 6-15: reserved

#### Server's RMB virtual address

The virtual address of the server's RMB as assigned by the server's RNIC.

Server's initial packet sequence number

The starting packet sequence number that this peer will use when sending to the other peer, so that the other peer can prepare its QP for the sequence number to expect.

#### A.2.4. SMC Confirm CLC message format

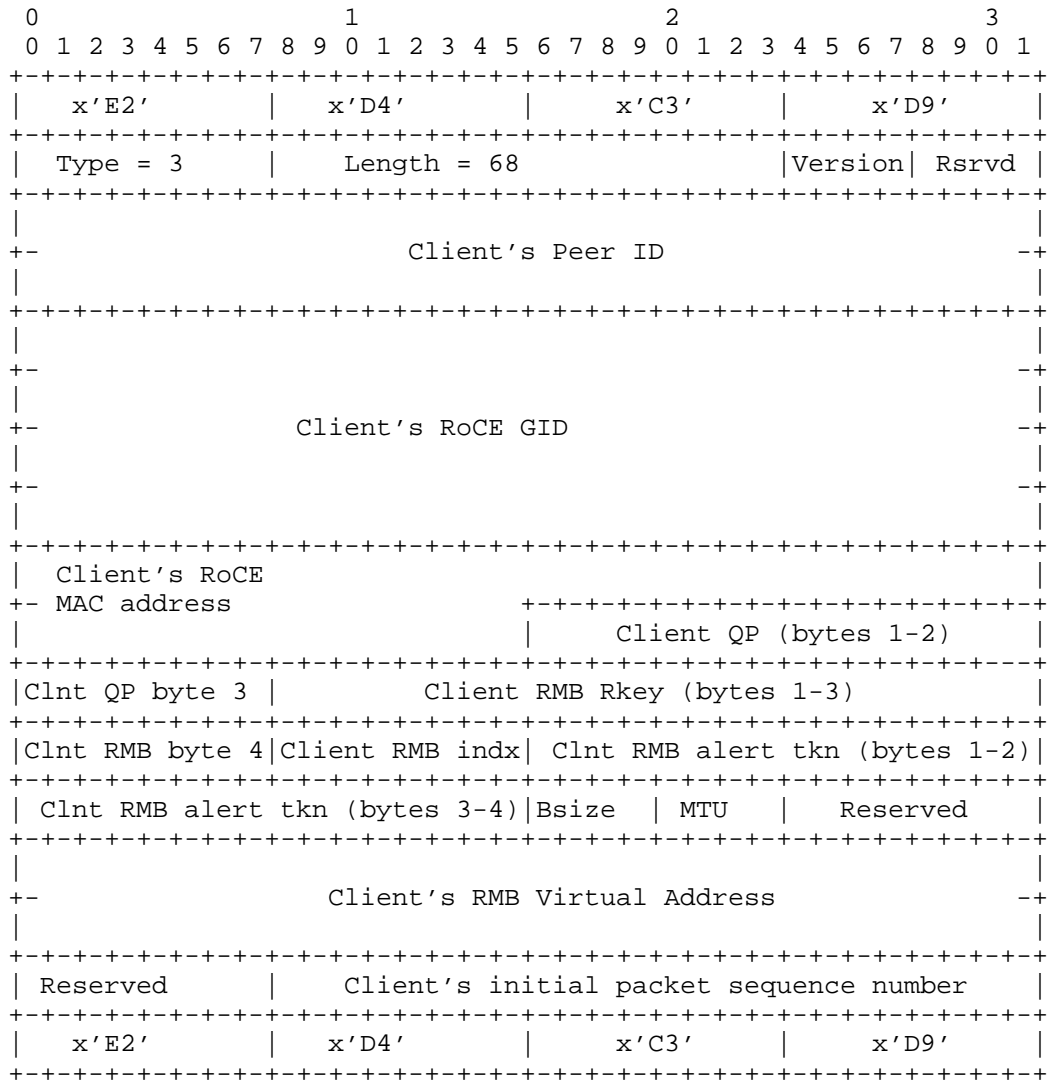


Figure 29 SMC Confirm CLC message format

The SMC Confirm CLC message is nearly identical to the SMC Accept except that it contains client information and lacks a first contact flag.

The fields present on the SMC Confirm CLC message are:

#### Eyecatchers

Like all CLC messages, the SMC Confirm has beginning and ending eyecatchers to aid with verification and parsing. The hex digits spell 'SMCR' in IBM-1047 (EBCDIC)

#### Type

CLC message type 3 indicates SMC Confirm

#### Length

The SMC Confirm CLC message is 68 bytes long

#### Version

Version of the SMC-R protocol. Version 1 is the only currently defined value.

#### Client's Peer ID

As described in A.2.1. above

#### Clients's RoCE GID

This is the IPv6 address of the RNIC that the client chose for this SMC Link

#### Client's RoCE MAC address

The MAC address of the client's RNIC for the SMC link. It is required as some operating systems do not have neighbor discovery or ARP support for RoCE RNICs.

#### Client's QP number

The number for the reliably connected queue pair that the client created for this SMC link

#### Client's RMB Rkey

The RDMA Rkey for the RMB that the client created or chose for this TCP connection

Client's RMB element index

This indexes which element within the client's RMB will represent this TCP connection

Client's RMB element alert token

A platform defined, architecturally opaque token that identifies this TCP connection. Added by the server as immediate data on RDMA writes from the server to the client to inform the client that there is data for this connection to retrieve from the RMB element

Bsize:

Client's RMB element buffer size in four bits compressed notation:  $x=4$  bits. Actual buffer size value is  $(2^{(x+4)}) * 1K$ . Smallest possible value is 16K. Largest size supported by this architecture is 512K.

MTU

An enumerated value indicating this peer's QP MTU size. The two peers exchange this value and the minimum of the peer's value will be used for the QP. The values are enumerated in A.2.3. This value should only be validated on the first contact exchange.

Client's RMB virtual address

The virtual address of the server's RMB as assigned by the server's RNIC.

Client's initial packet sequence number

The starting packet sequence number that this peer will use when sending to the other peer, so that the other peer can prepare its QP for the sequence number to expect

.

#### A.2.5. SMC Decline CLC message format

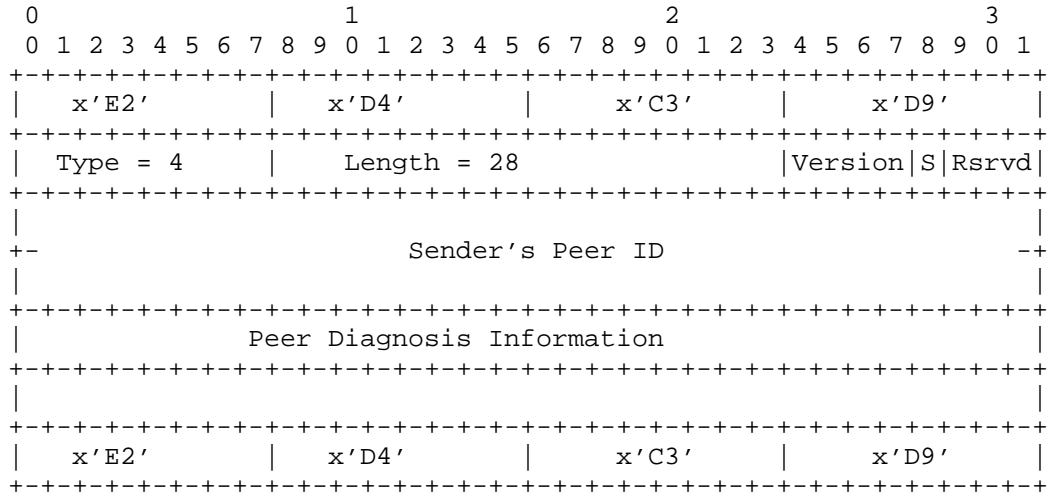


Figure 30 SMC Decline CLC message format

The fields present on the SMC Decline CLC message are:

##### Eyecatchers

Like all CLC messages, the SMC Decline has beginning and ending eyecatchers to aid with verification and parsing. The hex digits spell 'SMCR' in IBM-1047 (EBCDIC)

##### Type

CLC message type 4 indicates SMC Decline

##### Length

The SMC Decline CLC message is 28 bytes long

##### Version

Version of the SMC-R protocol. Version 1 is the only currently defined value.

##### S-bit

Synch Bit. Indicates that the link group is out of synch and receiving peer must clean up its representation of the link group

Sender's Peer ID

As described in A.2.1. above

Peer Diagnosis Information

Four bytes of diagnosis information provided by the peer. These values are defined by the individual peers and it is necessary to consult the peer's system documentation to interpret the results.

### A.3. LLC messages

LLC messages are sent over an existing SMC-R link using RoCE message passing and are always 44 bytes long so that they fit into the space available in a single WQE without requiring the receiver to post receive buffers. If all 44 bytes are not needed, they are padded out with zeroes. LLC messages are in a request/response format. The message type is the same for request and response, and a flag indicates whether a message is flowing as a request or a response.

The two high order bits of an LLC message opcode indicate how it is to be handled by a peer that does not support the opcode.

If the high order bits of the opcode are b'00' then the peer must support the LLC message and indicate a protocol error if it does not.

If the high order bits of the opcode are b'10' then the peer must silently discard the LLC message if does not support the opcode. This requirement is inserted to allow for toleration of advanced, but optional function.

High order bits of b'11' indicate a Connection Data Control (CDC) message as described in A.4.

## A.3.1. CONFIRM LINK LLC message format

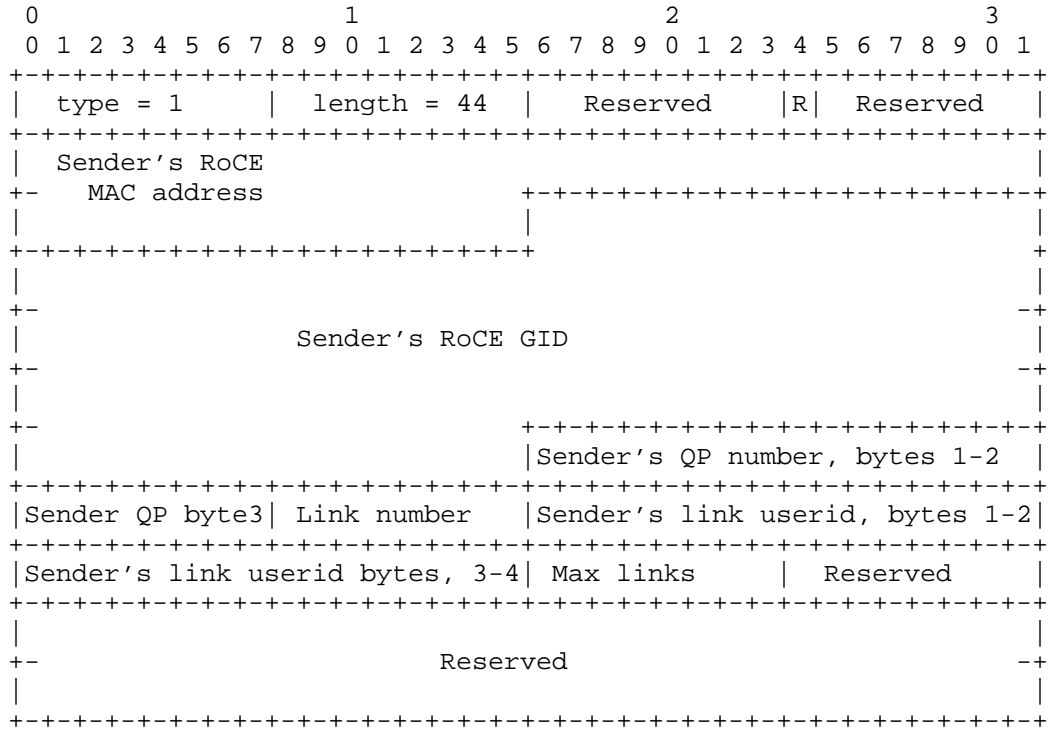


Figure 31 CONFIRM LINK LLC message format

The CONFIRM LINK LLC message is required to be exchanged between the server and client over a newly created SMC-R link to complete the setup of an SMC link. Its purpose is to confirm that the RoCE path is actually usable.

On first contact this flows after the server receives the SMC Confirm CLC message from the client over the IP connection. For additional links added to an SMC link group, it flows after the ADD LINK and ADD LINK CONTINUATION exchange. This flow provides confirmation that the queue pair is in fact usable. Each peer echoes its RoCE information back to the other.

Type

Type 1 indicates CONFIRM LINK

Length

All LLC messages are 44 bytes long

R

Reply flag. When set indicates this is a CONFIRM LINK REPLY

Sender's RoCE MAC address

The MAC address of the sender's RNIC for the SMC link. It is required as some operating systems do not have neighbor discovery or ARP support for RoCE RNICs.

Sender's RoCE GID

This is the IPv6 address of the RNIC that the sender is using for this SMC-R Link

Sender's QP number

The number for the reliably connected queue pair that the sender created for this SMC-R link

Link number

An identifier assigned by the server that uniquely identifies the link within the link group. This identifier is ONLY unique within a link group. Provided by the server and echoed back by the client

Link User ID

An opaque, implementation defined identifier assigned by the sender and provided to the receiver solely for purposes of display, diagnosis, network management, etc. The link user ID should be unique across the sender's entire software space, including all link other link groups.

Max Links

The maximum number of links the sender can support in a link group. The maximum for this link group is the the smaller of the values provided by the two peers.

## A.3.2. ADD LINK LLC message format

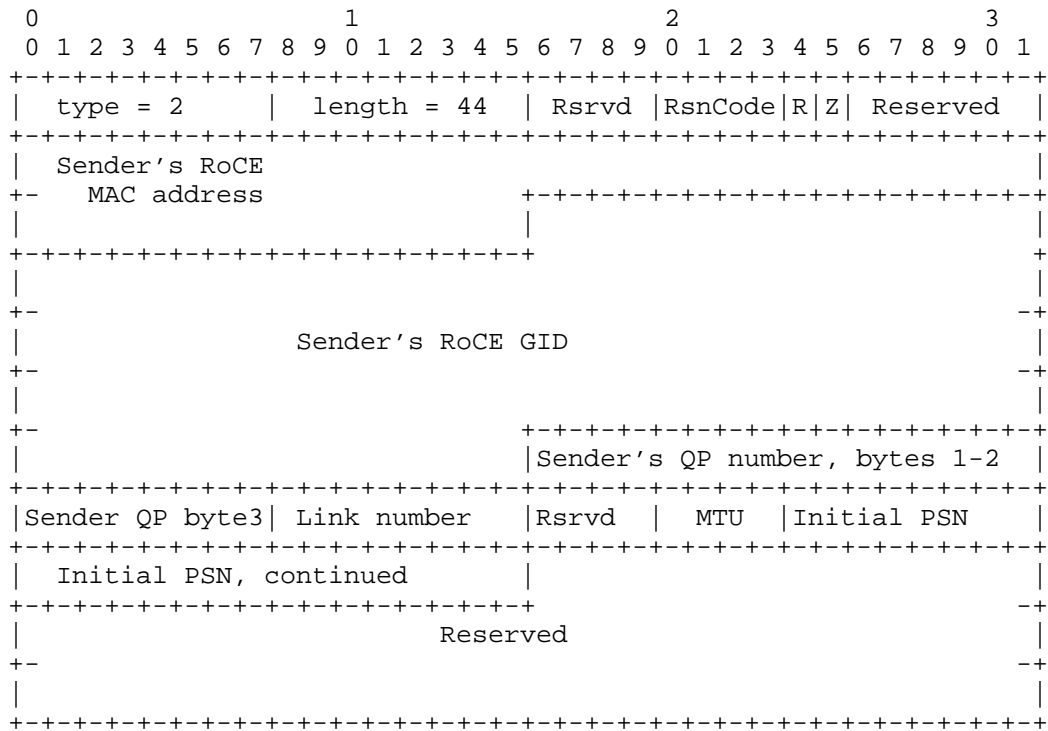


Figure 32 ADD LINK LLC message format

The ADD LINK LLC message is sent over an existing link in the link group when a peer wishes to add an SMC-R link to an existing SMC-R link group. It sent by the server to add a new SMC-R link to the group, or by the client to request that the server add a new link, for example when a new RNIC becomes active. When sent from the client to the server, it represents a request that the server initiate an ADD LINK exchange.

This message is sent immediately after the initial SMC link in the group completes, as described in 3.5.1. First contact. It can also be sent over an existing SMC-R link group at any time as new RNICs are added and become available. Therefore there can be as few as 1 new RMB RTokens to communicate, or several. Rtokens will be communicated using ADD LINK CONTINUATION messages.

The contents of the ADD LINK LLC message are:

Type

Type 2 indicates ADD LINK

Length

All LLC messages are 44 bytes long

RsnCode

If the Z (rejection) flag is set, this field provides the reason code. Values can be:

X'1' - no alternate path available: set when the server provides the same MAC/GID as an existing SMC-R link in the group, and the client does not have any additional RNICs available (i.e., server is attempting to set up an asymmetric link but none is available)

X'2' - Invalid MTU value specified

R

Reply flag. When set indicates this is an ADD LINK REPLY

Z

Rejection flag. When set on reply indicates that the server's ADD LINK was rejected by the client. When this flag is set, the reason code will also be set.

Sender's RoCE MAC address

The MAC address of the sender's RNIC for the new SMC-R link. It is required as some operating systems do not have neighbor discovery or ARP support for RoCE RNICs.

Sender's RoCE GID

The IPv6 address of the RNIC that the sender is using for the new SMC-R Link

Sender's QP number

The number for the reliably connected queue pair that the sender created for the new SMC-R link

Link number

An identifier for the new SMC-R link. This is assigned by the server and uniquely identifies the link within the link group. This identifier is ONLY unique within a link group. Provided by the server and echoed back by the client

#### MTU

An enumerated value indicating this peer's QP MTU size. The two peers exchange this value and the minimum of the peer's value will be used for the QP. The values are enumerated in A.2.3.

#### Initial PSN

The starting packet sequence number that this peer will use when sending to the other peer, so that the other peer can prepare its QP for the sequence number to expect.

#### A.3.3. ADD LINK CONTINUATION LLC message format

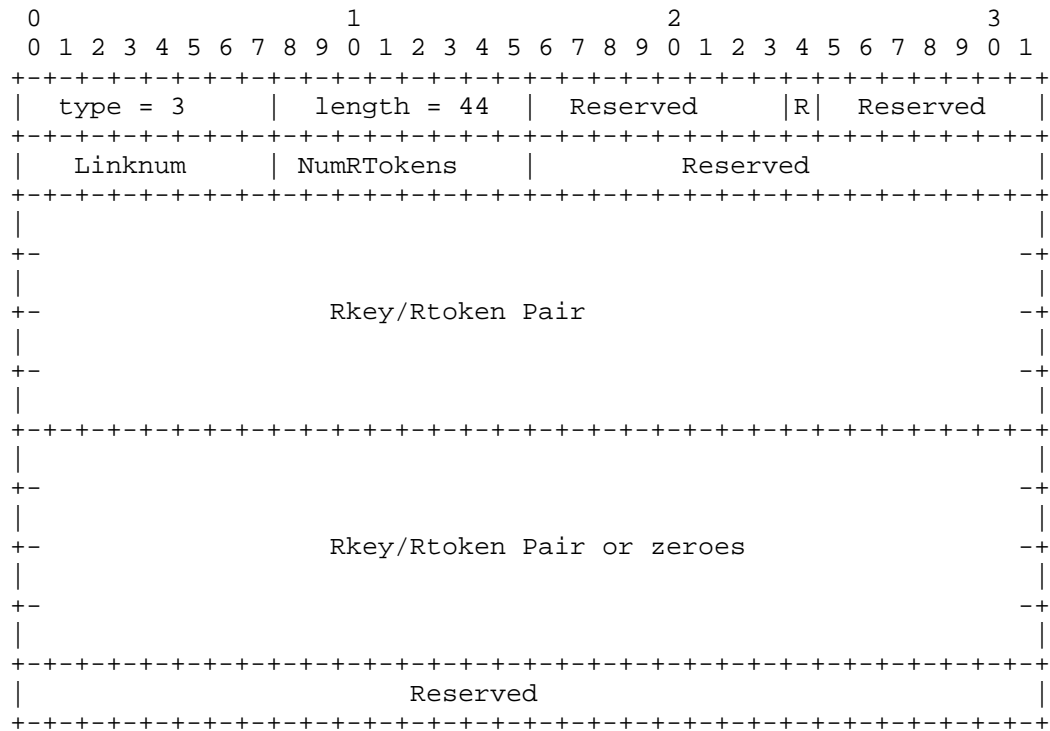


Figure 33 ADD LINK CONTINUATION LLC message format

When a new SMC-R link is added to an SMC-R link group, it is necessary to communicate the new link's RTokens for the RMBs that the SMC-r link group can access. This message follows the ADD LINK and provides the RTokens.

The server kicks off this exchange by sending the first ADD LINK CONTINUATION LLC message, and the server controls the exchange as described below.

- o If the client and the server require the same number of ADD LINK CONTINUATION messages to communicate their RTokens, the server starts the exchange by sending the client the first ADD LINK CONTINUATION request to the client with its RTokens, then the client responds with an ADD LINK CONTINUATION response with its RTokens, and so on until the exchange is completed.
- o If the server requires more ADD LINK CONTINUATION messages than the client, then after the client has communicated all its RTokens, the server continues to send ADD LINK CONTINUATION request messages to the client. The client continues to respond, using empty (number of RTokens to be communicated = 0) ADD LINK CONTINUATION response messages.
- o If the client requires more ADD LINK CONTINUATION messages than the server, then after communicating all its RTokens the server will continue to send empty ADD LINK CONTINUATION messages to the client to solicit replies with the client's RTokens, until all have been communicated.

The contents of this message are:

Type

Type 3 indicates ADD LINK CONTINUATION

Length

All LLC messages are 44 bytes long

R

Reply flag. When set indicates this is an ADD LINK CONTINUATION  
REPLY

## LinkNum

The link number of the new link within the SMC link group that Rkeys are being communicated for

## NumRTokens

Number of RTokens remaining to be communicated (including the ones in this message). If the value is less than or equal to 2, this is the last message. If it is greater than 2, another continuation message will be required, and its value will be the value in this message minus 2, and so on until all Rkeys are communicated. The maximum value for this field is 255.

Up to 2 Rkey/RToken pairs

These consist of an Rkey for an RMB that is known on the SMC-R link that this message was sent over (the reference Rkey), paired with the same RMB's RToken over the new SMC link. A full RToken is not required for the reference because it is only being used to distinguish which RMB it applies to, not address it.

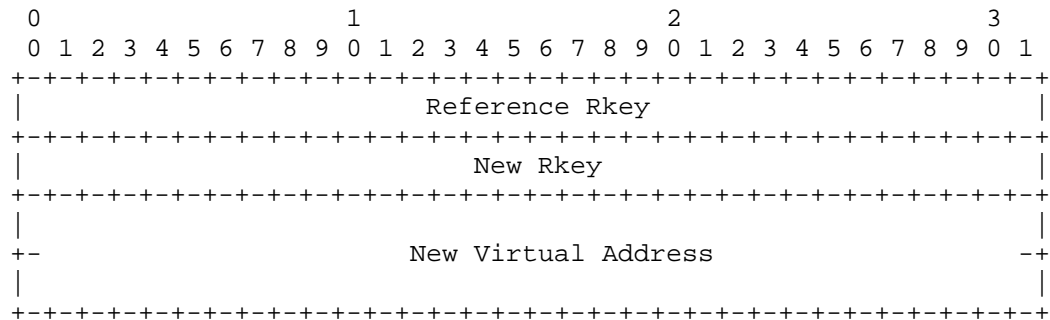


Figure 34 Rkey/Rtoken pair format

The contents of the RKey/RToken pair are:

## Reference Rkey

The Rkey of the RMB as it is already known on the SMC-R link over which this message is being sent. Required so that the peer knows which RMB to associate the new Rtoken with.

New Rkey

The Rkey of this RMB as it is known over the new SMC-R link



The server can also initiate the DELETE Link without notification from the client if it detects an error or if orderly link termination was initiated.

The client may also request termination of the entire link group and the server may terminate the entire link group using this message.

The contents of this message are:

Type

Type 4 indicates DELETE LINK

Length

All LLC messages are 44 bytes long

R

Reply flag. When set indicates this is an DELETE LINK REPLY

A

All flag. When set indicates that all links in the link group are to be terminated. This terminates the link group.

O

Orderly flag. Indicates orderly termination. Orderly termination is generally caused by an operator command rather than an error on the link. When the client requests orderly termination, the server may wait to complete other work before terminating.

LinkNum

The link number of the link to be terminated. If the A flag is set, this field has no meaning and is set to 0.

RsnCode

The termination reason code. Currently defined reason codes are:

Request Reason Codes:

- o X'00010000' = lost path
- o X'00020000' = operator initiated termination

- o X'00030000' = Program initiated termination (link inactivity)
- o X'00040000' = LLC protocol violation
- o X'00050000' = Asymmetric link no longer needed

Response Reason Codes:

- o X'00100000' = Unknown Link ID (no link)
- o Others TBD

#### A.3.5. CONFIRM RKEY LLC message format

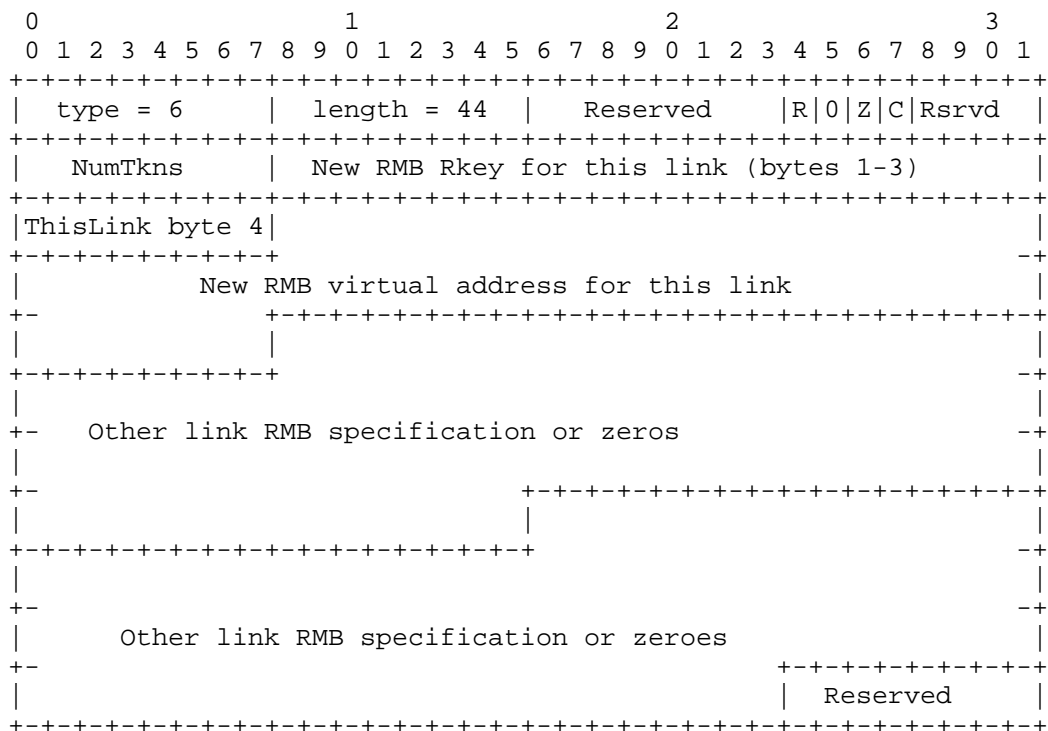


Figure 36 CONFIRM RKEY LLC message format

The CONFIRM\_RKEY flow can be sent at any time from either the client or the server, to inform the peer that an RMB has been created or deleted. The creator of a new RMB must inform its peer of the new RMB's RToken for all SMC-R links in the SMC-R link group. The deleter of an RMB must inform its peer of the deleted RMB's RToken for all SMC-R links.

For RMB creation, the creator sends this message over the SMC link that the first TCP connection that uses the new RMB is using. This message contains the new RMB RToken for the SMC link that the message is sent over, then it lists the sender's SMC links in the link group paired with the new RToken for the new RMB for that link. This message can communicate the new RTokens for 3 QPs: the QP for the link this message is sent over, and 2 others. If there are more than 3 links in the SMC-R link group, CONFIRM\_RKEY\_CONTINUATION will be required.

For RMB deletion, the creator sends the same format of message with a delete flag set, to inform the peer that the RMB's RTokens on all links in the group are deleted.

In both cases, the peer responds by simply echoing the message with the response flag set. If the response is a negative response, the sender must recalculate the RToken set and start a new CONFIRM\_RKEY exchange from the beginning. The timing of this retry is controlled by the C flag as described below.

The contents of this message are:

Type

Type 6 indicates CONFIRM RKEY

Length

All LLC messages are 44 bytes long

R

Reply flag. When set indicates this is a CONFIRM RKEY REPLY

0

Reserved bit

Z

Negative response flag

C

Configuration Retry bit. If this is a negative response and this flag is set, the originator should recalculate the Rkey set and retry this exchange as soon as the current configuration change

is completed. If this flag is not set on a negative response, the originator must wait for the next natural stimulus (for example, a new TCP connection started that requires a new RMB) before retrying.

#### NumTkns

The number of other link/RToken pairs, including those provided in this message, to be communicated. Note that this value does not include the Rtoken for the link this message was sent on (i.e., the maximum value is 2). If this value is three or fewer this is the only message in the exchange. If this value is greater than three, a CONFIRM RKEY CONTINUATION message will be required.

Note: in this version of the architecture, 8 is the maximum number of links supported in a link group.

#### New RMB Rkey for this link

The new RMB's Rkey as assigned on the link this message is being sent over.

#### New RMB virtual address for this link

The new RMB's virtual address as assigned on the link this messages is being sent over.

#### Other link RMB specification

The new RMB's specification on the other links in the link group, as shown in Figure 38.

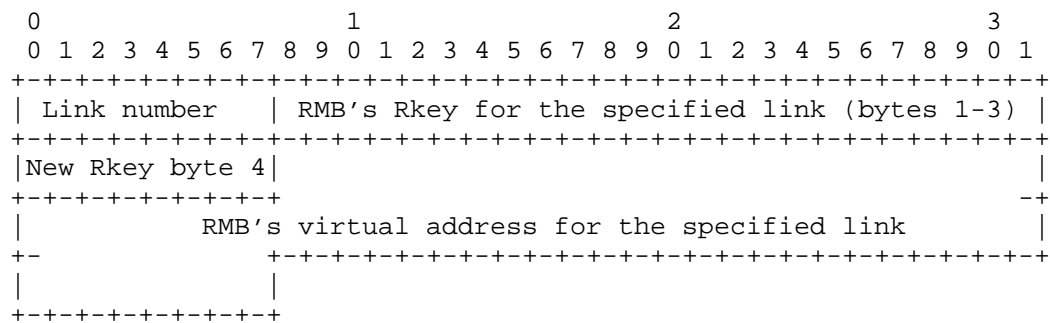


Figure 37 Format of link number/Rkey pairs



including the ones in this message. If the value is 3 or less, this is the last message of the group. If the value is 4 or higher, additional CONFIRM RKEY CONTINUATION messages will follow, and the Numlinks value will be a countdown until all are communicated.

Like the CONFIRM RKEY message, the peer responds by echoing the message back with the reply flag set.

The contents of this message are:

Type

Type 8 indicates CONFIRM RKEY CONTINUATION

Length

All LLC messages are 44 bytes long

R

Reply flag. When set indicates this is a CONFIRM RKEY CONTINUATION REPLY

O

Reserved bit

Z

Negative response flag

NumTknsLeft

The number of link/RToken pairs, including those provided in this message, that are remaining to be communicated. If this value is three or fewer this is the last message in the exchange. If this value is greater than three, another CONFIRM RKEY CONTINUATION message will be required. Note that in this version of the architecture, 8 is the maximum number of links supported in a link group.

Other link RMB specifications

The new RMB's specification on other links in the link group, as shown in Figure 38.

## A.3.7. DELETE RKEY LLC message format

0	1	2	3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1			
type = 9	length = 44	Reserved	R 0 Z  Rsrvd
Count	Error Mask	Reserved	
First deleted Rkey			
Second deleted Rkey or zeros			
Third deleted Rkey or zeros			
Fourth deleted Rkey or zeros			
Fifth deleted Rkey or zeros			
Sixth deleted Rkey or zeros			
Seventh deleted Rkey or zeros			
Eighth deleted Rkey or zeros			
Reserved			

The DELETE\_RKEY flow can be sent at any time from either the client or the server, to inform the peer that one or more RMBs have been deleted. Because the peer already knows every RMB's Rkey on each link in the link group, this message only specifies one Rkey for each RMB being deleted. The Rkey provided for each deleted RMB will be its Rkey as known on the SMC-R link that this message is sent over.

It is not necessary to provide the entire RToken. The Rkey alone is sufficient for identifying an existing RMB.

The peer responds by simply echoing the message with the response flag set. If the peer did not recognize an Rkey, a negative response flag will be set, however no aggressive recovery action beyond logging the error will be taken.

The contents of this message are:

Type

Type 9 indicates DELETE RKEY

Length

All LLC messages are 44 bytes long

R

Reply flag. When set indicates this is a DELETE RKEY REPLY

O

Reserved bit

Z

Negative response flag

Count

Number of RMBs being deleted by this message. Maximum value is 8

Error Mask

If this is a negative response, indicates which RMBs were not successfully deleted. Each bit corresponds to a listed RMB. So for example b'01010000' indicates that the second and fourth Rkeys weren't successfully deleted.

Deleted Rkeys

A list of Count Rkeys. Provided on the request flow and echoed back on the response flow. Each Rkey is valid on the link this message is sent over, and represents a deleted RMB. Up to eight RMBs can be deleted in this message.

#### A.3.8. TEST LINK LLC message format

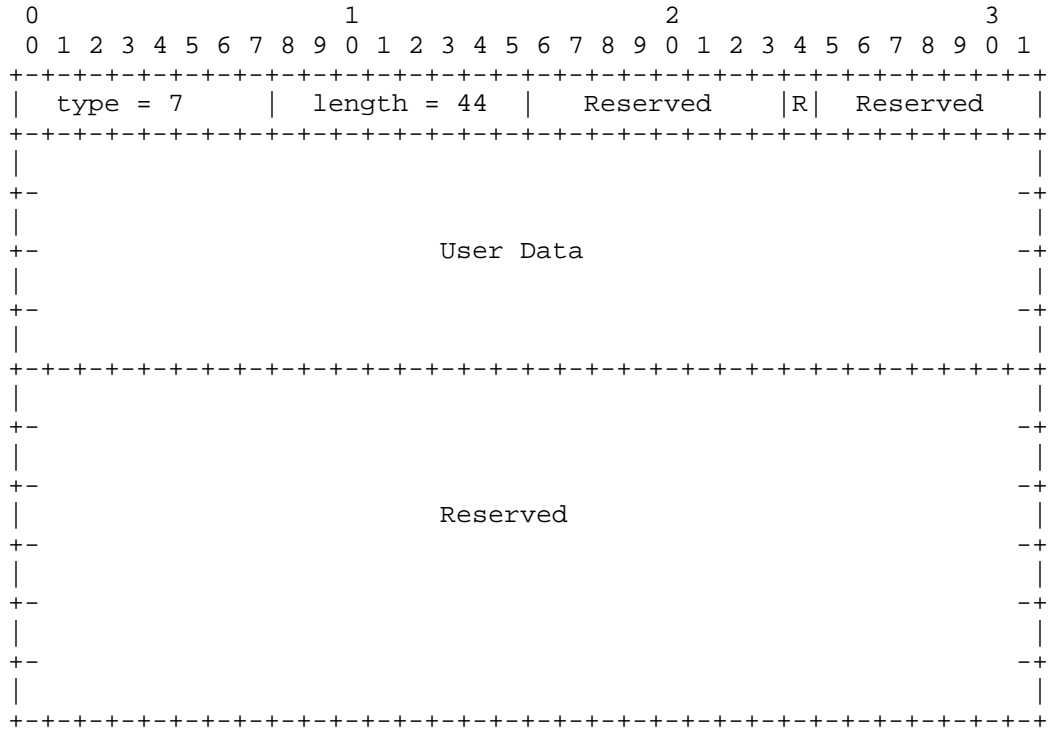


Figure 38 TEST LINK LLC message format

The TEST\_LINK request can be sent from either peer to the other on an existing SMC-R link at any time to test that the SMC-R link is active and healthy at the software level. A peer which receives a TEST\_LINK LLC message immediately sends back a TEST\_LINK reply, echoing back the user data. Also refer to 4.5.3. TCP Keepalive processing.

The contents of this message are:

Type

Type 7 indicates TEST LINK

Length

All LLC messages are 44 bytes long

R

Reply flag. When set indicates this is a TEST LINK REPLY

User Data

The receiver of this message echoes the sender's data back in a TEST\_LINK response LLC message

#### A.4. Connection Data Control (CDC) message format

The RMBE control data is communicated using Connection Data Control (CDC) messages, which use RDMA message passing using inline data, similar to LLC messages. Also similar to LLC messages, this data block is 44 bytes long to ensure that it can fit into private data areas of receive WQEs, without requiring the receiver to post receive buffers.

Unlike LLC messages, this data is integral to the data path so its processing must be prioritized and optimized similarly to other data path processing. While LLC messages may be processed on a slower path than data, these messages cannot be.

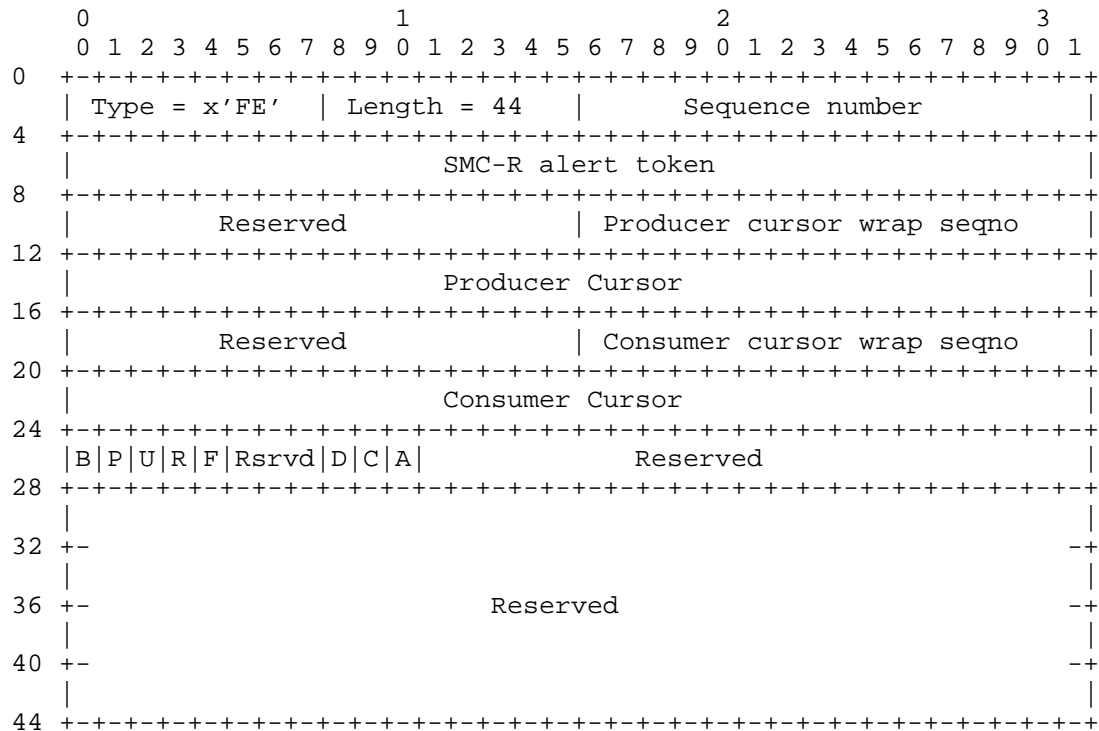


Figure 39 Connection Data Control (CDC) Message Format

Type = x'FE'

This type number has the two high order bits turned on to enable processing to quickly distinguish it from an LLC message

Length = 44

The length of inline data that does not require posting of a receive buffer.

Sequence number

A 2 byte unsigned integer that represents a wrapping sequence number. The initial value is one and this value can wrap to 0. Incremented with every control message send, except for the failover data validation message, and used to guard against processing an old control message out of sequence, and also used in failover data validation. In normal usage, if this number is

less than the last received value, discard this message. If greater, processes this message. Old control messages can be lost with no ill effect, but cannot be processed after newer ones.

If this is a failover validation CDC message (F flag set), then the receiver must verify that it has received and fully processed the RDMA write that was described by the CDC message with the sequence number in this message. If not, the TCP connection must be reset, to guard against data loss. Details of this processing are in section 4.6.1.

#### SMC-R alert token

The endpoint-assigned alert token that identifies which TCP connection on the link group this control message refers to.

#### Producer cursor wrap seqno

A 2 byte unsigned integer that represents wrapping counter incremented by the producer whenever the data written into this RMBE receiver buffer causes a wrap (i.e. the producer cursor wraps). This is used by the receiver to determine when new data is available even though the cursors appear unchanged such as when a full window size write is completed (Producer cursor of this RMBE sent by peer = Local Consumer Cursor) or in scenarios where the Producer Cursor sent for this RMBE < Local Consumer Cursor).

#### Producer cursor

Unsigned, 4 byte integer that is a wrapping offset into the RMBE data area. Points to the next byte of data to be written by the sender. Can advance up to the receiver's Consumer Cursor as known by the sender. When the urgent data present indicator is on then points one byte beyond the last byte of urgent data. When computing this cursor, the presence of the eyecatcher in the RMBE data area must be accounted for. The first writeable data location in the RMBE is at offset 4, so this cursor begins at 4 and wraps to 4.

#### Consumer cursor wrap seqno

2 byte unsigned integer that mirrors the value of the Producer cursor wrap sequence number when the last read from this RMBE occurred. Used as an indicator on how far along the consumer is in reading data (i.e. processed last wrap point or not). The

producer side can use this indicator to detect whether more data can be written to the partner in full window write scenarios (where the Producer Cursor = Consumer Cursor as known on the remote RMBE). In this scenario if the consumer sequence number equals the local producer sequence number the producer knows that more data can be written.

#### Consumer Cursor

Unsigned 4 byte integer that is a wrapping offset into the sender's RMBE data area. Points to the offset of the next byte of data to be consumed by the peer in its own RMBE. When computing this cursor, the presence of the eyecatcher in the RMBE data area must be accounted for. The first writeable data location in the RMBE is at offset 4, so this cursor begins at 4 and wraps to 4. The sender cannot write beyond this cursor into the peer's RMBE without causing data loss.

#### B-bit

Writer blocked indicator: Sender is blocked for writing, requires explicit notification when receive buffer space is available.

#### P-bit

Urgent data pending: Sender has urgent data pending for this connection

#### U-bit

Urgent data present: Indicates that urgent is data present in the RMBE data area, and the producer cursor points to one byte beyond the last byte of urgent data.

#### R-bit

Request for consumer cursor update: Indicates that a consumer cursor update is requested bypassing any window size optimization algorithms.

#### F-bit

Failover validation indicator: sent by a peer to guard against data loss during failover when the TCP connection is being moved to another SMC-R link in the link group. When this bit is set the only other fields in the CDC message that are significant are the type, length, SMC-R alert token and the sequence number. The

receiver must validate that it has fully processed the RDMA write described by the previous CDC message bearing the same sequence number as this validation message. If it has, no further action is required. If it has not, the TCP connection must be reset. This processing is described in detail in section 4.6.1.

#### D-bit

Sending done indicator: Sent by a peer when it is done writing new data into the receiver's RMBE data area.

#### C-bit

Peer Closed Connection indicator: Sent by a peer when it is completely done with this connection and will no longer be making any updates to the receiver's RMBE, and will also not be sending any more control messages.

#### A-bit

Abnormal Close indicator: Sent by a peer when the connection is abnormally terminated (for example, the TCP connection was Reset). When sent it indicates that the peer is completely done with this connection and will no longer be making any updates to this RMBE or sending any more control messages. It also indicates that the RMBE owner must flush any remaining data on this connection and surface an error return code to any outstanding socket APIs on this connection (same processing as receiving an RST segment on a TCP connection).

## Appendix B.

## Socket API considerations

A key design goal for SMC-R is to require no application changes for exploitation. It is confined to socket applications using stream (i.e. TCP protocol) sockets over IPv4 or IPv6. By virtue of the fact that the switch to the SMC-R protocol occurs after a TCP connection is established no changes are required in socket address family or in the IP addresses and ports that the socket application are using. Existing socket APIs that allow the application to retrieve local and remote socket address structures for an established TCP connection (for example, `getsockname()` and `getpeername()`) will continue to function as they have before. Existing DNS setup and APIs for resolving hostnames to IP addresses and vice versa also continue to function without any changes. In general all of the usual socket APIs that are used for TCP communicates (send APIs, recv APIs, etc.) will continue to function as they do today even if SMC-R is used as the underlying protocol.

Each SMC-R enabled implementation does however need to pay special attention to any socket APIs that have a reliance on the underlying TCP and IP protocols and ensure that their behavior in an SMC-R environment is reasonable and minimizes impact to the application. While the basic socket API set is fairly similar across different Operating Systems, when it comes to advanced socket API options there is more variability. Each implementation needs to perform a detailed analysis of its API options and SMC-R impact and implications. As part of that step a discussion or review with other implementations supporting SMC-R would be useful to ensure a consistent implementation.

`setsockopt()/getsockopt()` considerations

These APIs allow socket applications to manipulate socket, transport (TCP/UDP) and IP level options associated with a given socket. Typically, a platform restricts the number of IP options available to stream (TCP) socket applications given their connection oriented nature. The general guideline here is to continue processing these APIs in a manner that allows for application compatibility. Some options will be relevant to the SMC-R protocol and will require special processing under the covers. For example, the ability to manipulate TCP send and receive buffer sizes is still valid for SMC-R. However, other options may have no meaning for SMC-R. For example, if an application enabled the `TCP_NODELAY` option to disable Nagle's algorithm it should have no real effect in SMC-R communications as there is no notion of Nagle's algorithm with this new protocol. But the implementation must accept the `TCP_NODELAY` option as it does today and save it so that it can be later extracted

via `getsockopt()` processing. Note that any TCP or IP level options will still have an effect on any TCP/IP packets flowing for an SMC-R connection (i.e. as part of TCP/IP connection establishment and TCP/IP connection termination packet flows).

Under the covers manipulation of the TCP options will also include the SMC layer setting and reading the SMC-R experimental option before and after completion of the 3 way TCP handshake.

## Appendix C. Rendezvous Error scenarios

Error scenarios in setting up and managing SMC-R links are discussed in this section.

### C.1. SMC Decline during CLC negotiation

A peer to the SMC-R CLC negotiation can send SMC Decline in lieu of any expected CLC message to decline SMC and force the TCP connection back to IP fabric. There can be several reasons for an SMC Decline during the CLC negotiation including: RNIC went down, SMC-R forbidden by local policy, subnet (IPv4) or prefix (IPv6) doesn't match, lack of resources to perform SMC-R. In all cases when an SMC Decline is sent in lieu of an expected CLC message, no confirmation is required and the TCP connection immediately falls back to using the IP fabric.

To prevent ambiguity between CLC messages and application data, an SMC Decline cannot "chase" another CLC message. SMC Decline can only be sent in lieu of an expected CLC message. For example, if the client sends SMC Proposal then its RNIC goes down, it must wait for the SMC Accept for the server and then it can reply to that with an SMC Decline.

This "no chase" rule means that if this TCP connection is not a first contact between RoCE peers, a server cannot send SMC Decline after sending SMC Accept - it can only either break the TCP connection. Similarly, once the client sends SMC Confirm on a TCP connection that isn't first contact, it is committed to SMC-R for this TCP connection and cannot fall back to IP.

### C.2. SMC Decline during LLC negotiation

For a TCP connection that represents first contact between RoCE pairs, it is possible for SMC to fail back to IP during the LLC negotiation. This is possible until the first contact SMC link is confirmed. For example, see Figure 40. After a first contact SMC link is confirmed, fallback to IP is no longer possible. The rule that this translates to is: a first contact peer can send SMC Decline at any time during LLC negotiation until it has successfully sent its CONFIRM LINK (request or response) flow. After that point, it cannot fall back to IP.

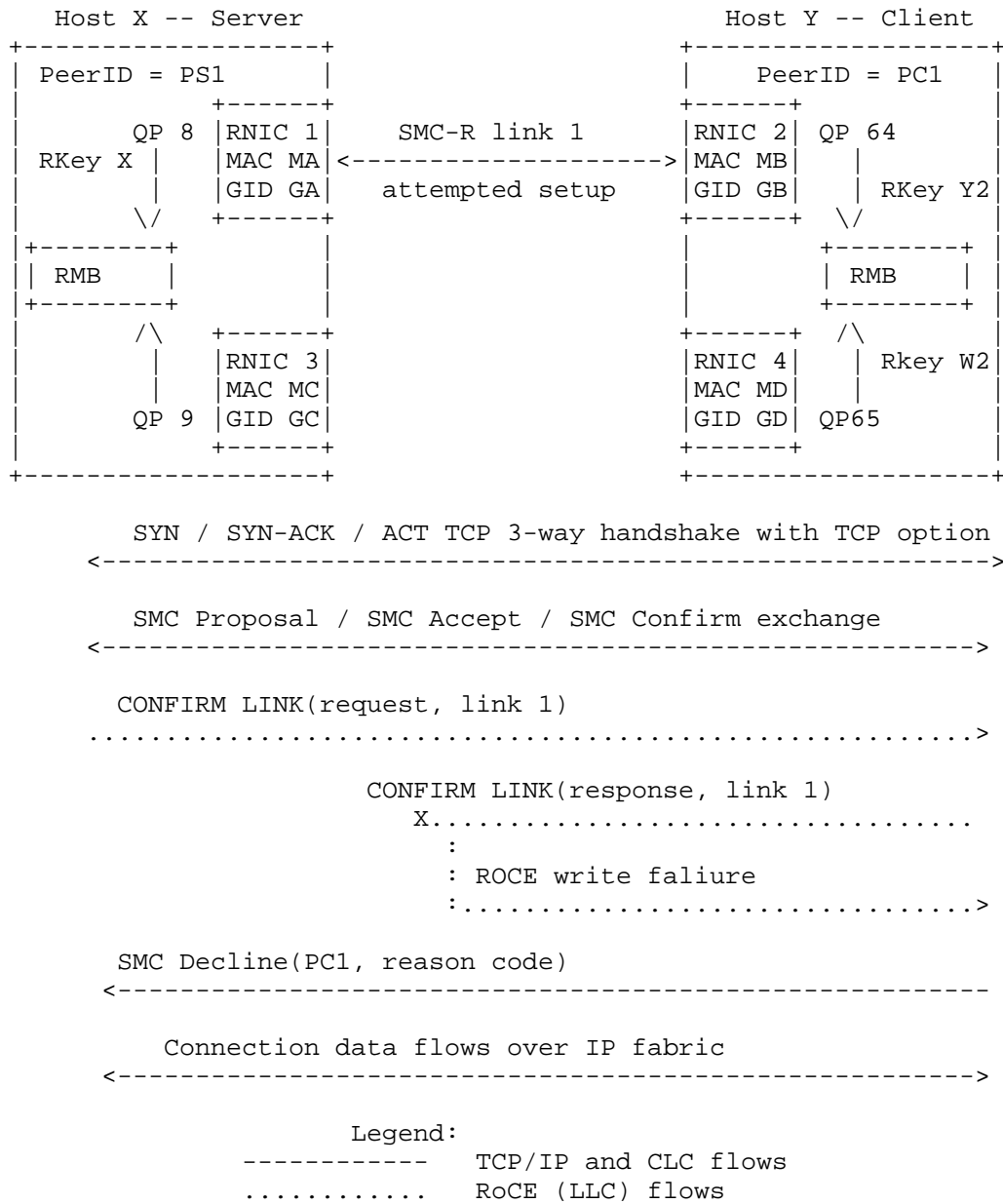


Figure 40 SMC Decline during LLC negotiation

### C.3. The SMC Decline window

Because SMC-R does not support fall-back to IP for a TCP connection that is already using RDMA, there are specific rules on when SMC Decline, which signals a fall-back to IP because of an error or problem with the RoCE fabric, can be sent during TCP connection setup. There is a point of no return after which a connection cannot fall back to IP, and RoCE errors that occur after this point require the connection to be broken with a RST flow in the IP fabric.

For first contact, that point of no return is after the Add Link LLC message has been successfully sent for the second SMC-R link. Specifically, the server cannot fall back to IP after receiving either a positive write completion indication for the Add Link request, or after receiving the Add Link response from the client, whichever comes first. The client cannot fall back to IP after either sending a negative Add Link response, receiving a positive write complete on a positive Add Link response, or receiving a Confirm Link for the second SMC-R link from the server, whichever comes first.

For subsequent contact, that point of no return is after the last send of the CLC negotiation completes. This, in combination with the rule that error "chasers" are not allowed during CLC negotiation, means that the server cannot send SMC Decline after sending an SMC Accept, and the client cannot send an SMC Decline after sending an SMC Confirm.

### C.4. Out of synch conditions during SMC-R negotiation

The SMC Accept CLC message contains a "first contact" flag that indicates to the client whether or not the server believes it is setting up a new link group, or using an existing link group. This flag is used to detect an out of synch condition between the client and the server. The scenario detected is as follows: There is a single existing SMC-R link between the peers. After the client sends the SMC Proposal CLC message, the existing SMC-R link between the client and the server fails. The client cannot chase the SMC Proposal CLC message with an SMC Decline CLC message in this case because the client does not yet know that the server would have wanted to choose the SMC-R link that just crashed. The QP that failed recovers before the server returns its SMC Accept CLC message. This means that there is a QP but no SMC link. Since the server had not yet learned of the SMC link failure when it sent the SMC Accept CLC message, it attempts to re-use the SMC link that just failed. This means the server would not set the "first contact" flag, indicating to the client that the server thinks it is reusing an SMC-

R link. However the client does not have an SMC-R link that matches the server's specification. Because the "first contact" flag is off, the client realizes it is out of synch with the server and sends SMC Decline to cause the connection to fall back to IP.

#### C.5. Timeouts during CLC negotiation

Because the SMC-R negotiation flows as TCP data, there are built-in timeouts and retransmits at the TCP layer for individual messages. Implementations also must protect the overall TCP/CLC handshake with a timer or timers to prevent connections from hanging indefinitely due to SMC-R processing. This can be done with individual timers for individual CLC messages or an overall timer for the entire exchange, which may include the TCP handshake and the CLC handshake under one timer or separate timers. This decision is implementation dependent.

If the TCP and/or CLC handshakes time out, the TCP connection must be terminated as it would be in a legacy IP environment when connection setup doesn't complete in a timely manner. Because the CLC flows are TCP messages, if they cannot be sent and received in a timely fashion, the TCP connection is not healthy and would not work if fallback to IP were attempted.

#### C.6. Protocol errors during CLC negotiation

Protocol errors occur during CLC negotiation when a message is received that is not expected. For example, a peer that is expecting a CLC message but instead receives application data has experienced a protocol error, and also indicates a likely software error as the two sides are out of synch. When application data is expected, this data is not parsed to ensure it's not a CLC message.

When a peer is expecting a CLC negotiation message, any parsing error except a bad enumerated value in that message must be treated as application data. The CLC negotiation messages are designed with beginning and ending eyecatchers to help verify that they are actually the expected message. If other parsing errors in an expected CLC message occur, such as incorrect length fields or incorrectly formatted fields, the message must be treated as application data.

All protocol errors with the exception of bad enumerated values must result in termination of the TCP connection. No fallback to IP is allowed in the case of a protocol error because if the protocols are out of synch, mismatched, or corrupted, then data and security integrity cannot be ensured.

The exception to this rule is enumerated values, for example the QP MTU values on SMC Accept and SMC Confirm. If a reserved value is received, the proper error response is to send SMC Decline and fall back to IP. The reason for this is that use of a reserved enumerated value indicates that the other partner likely has additional support that the receiving partner does not have. This indicated mismatch of SMC-R capabilities is not an integrity problem, but indicates that SMC-R cannot be used for this connection

#### C.7. Timeouts during LLC negotiation

Whenever a peer sends an LLC message to which a reply is expected, it sets a timer after the send posts to wait for the reply. An expected response may be a reply flavor of the LLC message (for example CONFIRM LINK REPLY) or a new LLC message (for example an ADD LINK CONTINUATION expected from the server by the client if there are more Rkeys to communicate).

On LLC flows that are part of a first contact setup of a link group, the value of the timer is implementation dependent but should be long enough to allow the other peer have a write complete timeout and 2-3 retransmits of an SMC Decline on the TCP fabric. For LLC flows that are maintaining the link group and not part of first contact setup of a link group, the timers may be shorter. Upon receipt of an expected reply the timer is cancelled. If a timer pops without a reply having been received, the sender must initiate a recovery action

During first contact processing, failure of an LLC verification timer is a should-not-occur which indicates a problem with one of the endpoints. The reason for this is that if there is a "routine" failure in the RoCE fabric that causes an LLC verification send to fail, the sender will get a write completion failure and will then send SMC Decline to the partner. The only time an LLC verification timer will expire on a first contact is when the sender thinks the send succeeded but it actually didn't. Because of the reliable connected nature of QP connections on the RoCE fabric, this indicates a problem with one of the peers, not with the RoCE fabric.

After the reliable connected QP for the first SMC-R link in a link group is set up on initial contact, the client sets a timer to wait for a RoCE verification message from the server that the QP is actually connected and usable. If the server experiences a failure sending its QP confirmation message, it will send SMC Decline, which should arrive at the client before the client's verification timer expires. If the client's timer expires without receiving either an SMC Decline or a RoCE message confirmation from the server, there is

a problem either with the server or with the TCP fabric. In either case the client must break the TCP connection and clean up the SMC-R link.

There are two scenarios in which the client's response to the QP verification message fails to reach the server. The main difference is whether or not the client has successfully completed the send of the CONFIRM LINK response.

In the normal case of a problem with the RoCE path, the client will learn of the failure by getting a write completion failure, before the server's timer expires. In this case, the client sends an SMC Decline CLC message to the server and the TCP connection falls back to IP.

If the client's send of the Confirmation message receives a positive return code but for some reason still does not reach the server, or the client's SMC Decline CLC message fails to reach the server after the client fails to send its RoCE confirmation message, then the server's timer will time out and the server must break the TCP connection by sending RST. This is expected to be a very rare case, because if the client cannot send its CONFIRM LINK RSP LLC message, the client should get a negative return code and initiate fallback to IP. A client receiving a positive return code on a send that fails to reach the server should be extremely rare.

#### C.7.1. Recovery actions for LLC timeouts and failures

The following table describes recovery actions for LLC timeouts. A write completion failure or other indication of failure to send on the send of the LLC command is treated the same as a timeout.

LLC Message: CONFIRM LINK from server (first contact, first link in the link group)

Timer waits for: CONFIRM LINK reply from client

Recovery action: Break the TCP connection by sending RST and clean up the link. The server should have received an SMC Decline from the client by now if the client had an LLC send failure.

LLC Message: CONFIRM LINK from server (first contact, second link in the link group)

Timer waits for: CONFIRM LINK reply from client

Recovery action: The second link was not successfully set up. Send DELETE LINK to the client. Connection data cannot flow in the first link in the link group, until the reply to this DELETE LINK is received, to prevent the peers from being out of synch on the state of the link group.

LLC Message: CONFIRM LINK from server (not first contact)

Timer Waits for: CONFIRM LINK reply from client

Recovery action: Clean up the new link and set a timer to retry. Send DELETE LINK to the client, in case the client has a longer timer interval, so the client can stop waiting

LLC Message: CONFIRM LINK REPLY from client (first contact)

Timer waits for: ADD LINK from server

Recovery action: Clean up the SMC-R link and break the TCP connection by sending RST over the IP fabric. There is a problem with the server. If the server had a send failure, it should have have sent SMC Decline by now.

LLC Message: ADD LINK from server (first contact)

Timer waits for: ADD LINK reply from client

Recovery action: Break the TCP connection with RST and clean up RoCE resources. The connection is past the point where the server can fall back to IP, and if the client had a send problem it should have sent SMC Decline by now.

LLC Message: ADD LINK from server (not first contact)

Timer waits for: ADD LINK reply from client

Recovery action: Clean up resources (QP, RMB keys, etc) for the new link and treat the link that the ADD LINK was sent over as if it had failed. If there is another link available to resend the ADD LINK and the link group still needs another link, retry the ADD LINK over another link in the link group.

LLC Message: ADD LINK REPLY from client (and there are more Rkeys to be communicated)

Timer waits for: ADD LINK CONTINUATION from server

Recovery action: Treat the same as ADD LINK timer failure

LLC Message: ADD LINK REPLY or ADD LINK CONTINUATION reply from client (and there are no more Rkeys to be communicated, for the second link in a first contact scenario)

Timer waits for: CONFIRM LINK from the server on the new link

Recovery action: The new link has failed to set up. Send DELETE LINK to the server. Do not consider the socket opened to the client application until receiving confirmation from the server in the form of a DELETE LINK request for this link and sending the reply (to prevent the partners from being out of synch on the state of the link group).

Set a timer to send another ADD LINK to the server if there is still an unused RNIC on the client side.

LLC Message: ADD LINK REPLY or ADD LINK CONTINUATION reply from the client (and there are no more Rkeys to be communicated)

Timer waits for: CONFIRM LINK from the server, over the new link

Recovery action: Send a DELETE LINK to the server for the new link, then clean up any resource allocated for the new link and set a timer to send ADD LINK to the server if there is still an unused RNIC on the client side. The new link has failed to set up, but the link that the ADD LINK exchange occurred over is unaffected.

LLC Message: ADD LINK CONTINUATION from server

Timer waits for: ADD LINK CONTINUATION REPLY from client

Recovery action: Treat the same as ADD LINK timer failure

LLC Message: ADD LINK CONTINUATION reply from client (first contact, and RMB count fields indicate that the server owes more ADD LINK CONTINUATION messages)

Timer waits for: ADD LINK CONTINUATION from the server

Recovery action: Clean up the SMC link and break the TCP connection by sending RST. There is a problem with the server.

If the server had a send failure, it should have have sent SMC Decline by now.

LLC Message: ADD LINK CONTINUATION reply from client (not first contact and RMB count fields indicate that the server owes more ADD LINK CONTINUATION messages)

Timer waits for: ADD LINK CONTINUATION from server

Recovery action: Treat as is if client detected link failure on the link the ADD LINK exchange is using. Send DELETE LINK to the server over another active link if one exists, otherwise clean up the link group.

LLC Message: DELETE LINK from client

Timer waits for: DELETE LINK request from server

Recovery action: If the scope of the request is to delete a single link, the surviving link, over which the client sent the DELETE LINK is no longer usable either. If this is the last link in the link group, end TCP connections over the link group by sending RST packets. If there are other surviving links in the link group, resend over a surviving link. Also send a DELETE LINK over a surviving link for the link that the client attempted to send the initial DELETE LINK message over. If the scope of the request is to delete the entire link group, try resending on other links in the link group until success is achieved. If all sends fail, tear down the link group and any TCP connections that exist on it.

LLC Message: DELETE LINK from server (scope: entire link group)

Timer waits for: Confirmation from the adapter that the message was delivered.

Recovery action: Tear down the link group and any TCP connections that exist over it.

LLC Message: DELETE LINK from server (scope: single link)

Timer waits for: DELETE LINK reply from the client

Recovery action: The link over which the client sent the DELETE LINK is no longer usable either. If this is the last link in the link group, end TCP connections over the link group by sending RST packets. If there are other surviving links in the link

group, resend over a surviving link. Also send a DELETE LINK over a surviving link for the link that the server attempted to send the initial DELETE LINK message over. If the scope of the request is to delete the entire link group, try resending on other links in the link group until success is achieved. If all sends fail, tear down the link group and any TCP connections that exist on it.

LLC Message: CONFIRM RKEY from the client

Timer waits for: CONFIRM RKEY REPLY from the server

Recovery action: Perform normal client procedures for detection of failed link. The link over which the message was sent has failed.

LLC Message: CONFIRM RKEY from the server

Timer waits for : CONFIRM RKEY REPLY from the client

Recovery action: Perform normal server procedures for detection of failed link. The link over which the message was sent has failed.

LLC Message: TEST LINK from the client

Timer waits for: TEST LINK REPLY from the server

Recovery action: Perform normal client procedures for detection of failed link. The link over which the message was sent has failed.

LLC Message: TEST LINK from the server

Timer waits for : TEST LINK REPLY from the client

Recovery action: Perform normal server procedures for detection of failed link. The link over which the message was sent has failed.

The following table describes recovery actions for invalid LLC messages. These could be misformatted or contain out of synch data.

LLC Message received: CONFIRM LINK from server

What could be bad: Incorrect link information

Recovery action: Protocol error. The link must be brought down by sending a DELETE LINK for the link over another link in the link group if one exists. If this is first contact, fall back to IP by sending SMC Decline to server.

LLC Message received: ADD LINK

What could be bad: Undefined enumerated MTU value

Recovery action: Send negative ADD LINK reply with reason code x'2'

LLC Message received: ADD LINK reply from client

What could be bad: Client side link information that would result in a parallel link being set up

Recovery action: Parallel links are not permitted. Delete the link by sending DELETE LINK to the client over another link in the link group.

LLC Message received: Any link group command from the server except DELETE LINK for the entire link group

What could be bad: Client has sent DELETE LINK for the link that the message was received on

Recovery action: Ignore the LLC message. Worst case the server will time out. Best case the DELETE LINK crosses with the command from the server and the server realizes it failed.

LLC Message received: ADD LINK CONTINUATION from the server or ADD LINK CONTINUATION REPLY from the client

What could be bad: Number of RMBs provided doesn't match count given on initial ADD LINK or ADD LINK reply message

Recovery action: Protocol error. Treat as if detected link outage

LLC Message received: DELETE LINK from client

What could be bad: Link indicated doesn't exist

Recovery action: If the link is in the process of being cleaned up, assume timing window and ignore message. Otherwise, send DELETE LINK REPLY with reason code 1.

LLC Message received: DELETE LINK from server

What could be bad: Link indicated doesn't exist

Recovery action: Send DELETE LINK REPLY with reason code 1.

LLC Message received: CONFIRM RKEY from either client or server

What could be bad: No Rkey provided for one or more of the links in the link group

Recovery action: Treat as if detected failure of the link(s) for which no RKEY was provided

LLC message received: DELETE RKEY

Specified RKey doesn't exist

Send negative DELETE RKEY response.

LLC message received: TEST LINK reply

What could be bad: User data doesn't match what was sent in the TEST LINK request

Recovery action: Treat as if detected that the link has gone down. This is a protocol error

LLC message received: Unknown LLC type with high order bits of opcode equal b'10'

What could be bad: This is an optional LLC message which the receiver does not support

Recovery action: Ignore (silently discard) the message

LLC message received: any unambiguously incorrect or out of synch LLC message

What it indicates: Link is out of sync

Recovery action: Treat as if detected that the link has gone down. Note that an unsupported or unknown LLC opcode whose two high order bits are b'10' is not an error, and must be silently discarded. Any other unknown or unsupported LLC opcode is an error.

#### C.8. Failure to add second SMC-R link to a link group

When there is any failure in setting up the second SMC-R link in an SMC-R link group, including confirmation timer expiration, the SMC-R link group is allowed to continue, without available failover. However this situation is extremely undesirable and the server must endeavor to correct it as soon as it can.

The server peer in the SMC-R link group must set a timer to drive it to retry setup of a failed additional SMC-R link. The server will immediately retry the SMC-R link setup when the first of the following events occurs:

- o The retry timer expires
- o A new RNIC becomes available to the server, on the same LAN as the SMC-R link group
- o An "Add Link" LLC request message is received from the client, which indicates availability of a new RNIC on the client side.

Authors' Addresses

Mike Fox  
IBM  
3039 Cornwallis Rd.  
Research Triangle Park, NC 27709  
  
Email: mjfox@us.ibm.com

Constantinos (Gus) Kassimis  
IBM  
3039 Cornwallis Rd.  
Research Triangle Park, NC 27709  
  
Email: kassimis@us.ibm.com

Jerry Stevens  
IBM  
3039 Cornwallis Rd.  
Research Triangle Park, NC 27709  
  
Email: sjerry@us.ibm.com



TCP Maintenance and Minor Extensions  
(tcpm)  
Internet-Draft  
Updates: 793 (if approved)  
Intended status: Standards Track  
Expires: September 30, 2012

F. Gont  
UTN-FRH / SI6 Networks  
March 29, 2012

Processing of TCP segments with Mirrored End-points  
draft-gont-tcpm-tcp-mirrored-endpoints-00.txt

## Abstract

This document describes a problem found in some popular implementations regarding the processing of TCP segments in which the local endpoint is equal to the remote endpoint. Additionally, it formally updates RFC 793 clarifying how this scenario should be handled.

## Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 30, 2012.

## Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Updating RFC 793 . . . . .	3
3. IANA Considerations . . . . .	3
4. Security Considerations . . . . .	3
5. Acknowledgements . . . . .	4
6. References . . . . .	4
6.1. Normative References . . . . .	4
6.2. Informative References . . . . .	4
Author's Address . . . . .	4

## 1. Introduction

Some systems have been found to be unable to process TCP segments in which the source endpoint {Source Address, Source Port} is the same than the destination end-point {Destination Address, Destination Port}. Such TCP segments have been reported to cause malfunction of a number of implementations [CERT1996], and have been exploited in the past to perform Denial of Service (DoS) attacks [Meltman1997]. While these packets are very very unlikely to exist in legitimate scenarios, TCP should nevertheless be able to process them without the need of any "extra" code.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 2. Updating RFC 793

TCP MUST be able to gracefully handle the case where the source end-point (IP Source Address, TCP Source Port) is the same as the destination end-point (IP Destination Address, TCP Destination Port).

A SYN segment in which the source end-point {Source Address, Source Port} is the same as the destination end-point {Destination Address, Destination Port} will result in a "simultaneous open" scenario, such as the one described in page 32 of RFC 793 [RFC0793]. Therefore, those TCP implementations that correctly handle simultaneous opens should already be prepared to handle these unusual TCP segments.

## 3. IANA Considerations

This document has no IANA actions. The RFC Editor is requested to remove this section before publishing this document as an RFC.

## 4. Security Considerations

This document describes a problem found in some popular implementations regarding the processing of TCP instances in which the local and the remote TCP endpoints are the equal. It formally updates RFC 793, clarifying how such packets should be handled, thus helping prevent unexpected behaviors in host implementations.

## 5. Acknowledgements

The author would like to thank David Borman for a fruitful discussion about this topic at IETF 73 (Minneapolis).

This document is based on the technical report "Security Assessment of the Transmission Control Protocol (TCP)" [CPNI-TCP] written by Fernando Gont on behalf of the UK CPNI.

Fernando Gont would like to thank the UK CPNI for their continued support.

## 6. References

### 6.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

### 6.2. Informative References

[CERT1996]  
CERT, "CERT Advisory CA-1996-21: TCP SYN Flooding and IP Spoofing Attacks", 1996,  
<<http://www.cert.org/advisories/CA-1996-21.html>>.

[CPNI-TCP]  
Gont, F., "CPNI Technical Note 3/2009: Security Assessment of the Transmission Control Protocol (TCP)", 2009, <<http://www.gont.com.ar/papers/tn-03-09-security-assessment-TCP.pdf>>.

[Meltman1997]  
Meltman, "new TCP/IP bug in win95. Post to the bugtraq mailing-list", 1996,  
<<http://insecure.org/sploits/land.ip.DOS.html>>.

Author's Address

Fernando Gont  
UTN-FRH / SI6 Networks  
Evaristo Carriego 2644  
Haedo, Provincia de Buenos Aires 1706  
Argentina

Phone: +54 11 4650 8472  
Email: fgont@si6networks.com  
URI: <http://www.si6networks.com>



TCP Maintenance and Minor Extensions  
(tcpm)  
Internet-Draft  
Updates: 793 (if approved)  
Intended status: Standards Track  
Expires: September 30, 2012

F. Gont  
UTN-FRH / SI6 Networks  
March 29, 2012

Processing of IP Security/Compartment and Precedence Information by TCP  
draft-gont-tcpm-tcp-seccomp-prec-00.txt

#### Abstract

This document discusses the security and interoperability problems that may arise as a result of the processing of IP security/compartment and precedence information by TCP. Additionally, it formally updates RFC 793 such that these issues are mitigated.

#### Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 30, 2012.

#### Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as

described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Updating RFC 793 . . . . .	4
3. IANA Considerations . . . . .	4
4. Security Considerations . . . . .	4
5. Acknowledgements . . . . .	4
6. References . . . . .	4
6.1. Normative References . . . . .	4
6.2. Informative References . . . . .	5
Author's Address . . . . .	5

## 1. Introduction

Section 3.9 (page 71) of RFC 793 [RFC0793] states that if the IP security/compartments and precedence of an incoming segment does not exactly match the security/compartments in the TCB, a RST segment should be sent, and the connection should be aborted.

A discussion of the IP security options relevant to this section can be found in Section 3.13.2.12, Section 3.13.2.13, and Section 3.13.2.14 of [RFC6274].

This certainly provides another attack vector for performing connection-reset attacks, as an attacker could forge TCP segments with a security/compartments that is different from that recorded in the corresponding TCB and, as a result, the attacked connection would be reset.

It is interesting to note that for connections in the ESTABLISHED state, this check is performed after validating the TCP Sequence Number and checking the RST bit, but before validating the Acknowledgement field. Therefore, even if the stricter validation of the Acknowledgement field (described in Section 3.4) was implemented, it would not help to mitigate this attack vector.

Resetting a connection due to a change in the Precedence value could also have a negative impact on interoperability. For example, the packets that correspond to a TCP connection could temporarily take a different internet path, in which some middle-box could re-mark the Precedence field (due to administration policies at the network to be transited). In such a scenario, an implementation following the advice in RFC 793 would abort the connection, when the connection would have otherwise probably survived.

While the IPv4 Type of Service field (and hence the Precedence field) has been redefined by the Differentiated Services (DS) field specified in RFC 2474 [RFC2474], RFC 793 [RFC0793] was never formally updated in this respect. We note that both legacy systems that have not been upgraded to implement the differentiated services architecture described in RFC 2475 [RFC2475] and current implementations that have extrapolated the discussion of the Precedence field to the Differentiated Services field may still be vulnerable to the connection reset vector discussed in Section 1.

Section 2 formally updates RFC 793 [RFC0793] such that these issues are mitigated.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this

document are to be interpreted as described in RFC 2119 [RFC2119].

## 2. Updating RFC 793

If the IP security/compartments field of an incoming TCP segment does not match the value recorded in the corresponding TCB, TCP MUST NOT abort the connection, but simply discard the corresponding packet. Additionally, this whole event SHOULD be logged as a security violation.

If the IP Differentiated Services field of an incoming TCP segment does not match the value recorded in the corresponding TCB, TCP MUST NOT abort the corresponding connection.

## 3. IANA Considerations

This document has no IANA actions. The RFC Editor is requested to remove this section before publishing this document as an RFC.

## 4. Security Considerations

This document discusses the processing of the IP security/compartments and precedence information, and the interoperability and security implications that arise from it. It updates RFC 793 such that the aforementioned issues are eliminated.

## 5. Acknowledgements

This document is based on the technical report "Security Assessment of the Transmission Control Protocol (TCP)" [CPNI-TCP] written by Fernando Gont on behalf of the UK CPNI.

Fernando Gont would like to thank the UK CPNI for their continued support.

## 6. References

### 6.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black,  
"Definition of the Differentiated Services Field (DS  
Field) in the IPv4 and IPv6 Headers", RFC 2474,  
December 1998.

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z.,  
and W. Weiss, "An Architecture for Differentiated  
Services", RFC 2475, December 1998.

## 6.2. Informative References

[CPNI-TCP]  
Gont, F., "CPNI Technical Note 3/2009: Security Assessment  
of the Transmission Control Protocol (TCP)", 2009, <[http://  
www.gont.com.ar/papers/  
tn-03-09-security-assessment-TCP.pdf](http://www.gont.com.ar/papers/tn-03-09-security-assessment-TCP.pdf)>.

[RFC6274] Gont, F., "Security Assessment of the Internet Protocol  
Version 4", RFC 6274, July 2011.

## Author's Address

Fernando Gont  
UTN-FRH / SI6 Networks  
Evaristo Carriego 2644  
Haedo, Provincia de Buenos Aires 1706  
Argentina

Phone: +54 11 4650 8472  
Email: [fgont@si6networks.com](mailto:fgont@si6networks.com)  
URI: <http://www.si6networks.com>



TCP Maintenance and Minor Extensions  
(tcpm)  
Internet-Draft  
Intended status: Experimental  
Expires: April 25, 2013

P. Hurtig  
Karlstad University  
A. Petlund  
Simula Research Laboratory AS  
M. Welzl  
University of Oslo  
October 22, 2012

TCP and SCTP RTO Restart  
draft-hurtig-tcpm-rtorestart-03

Abstract

This document describes a modified algorithm for managing the TCP and SCTP retransmission timers that provides faster loss recovery when a connection's amount of outstanding data is small. The modification allows the transport to restart its retransmission timer more aggressively in situations where fast retransmit cannot be used. This enables faster loss detection and recovery for connections that are short-lived or application-limited.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 25, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## 1. Introduction

TCP uses two mechanisms to detect segment loss. First, if a segment is not acknowledged within a certain amount of time, a retransmission timeout (RTO) occurs, and the segment is retransmitted [RFC6298]. While the RTO is based on measured round-trip times (RTTs) between the sender and receiver, it also has a conservative lower bound of 1 second to ensure that delayed segments are not mistaken as lost. Second, when a sender receives duplicate acknowledgments, the fast retransmit algorithm infers segment loss and triggers a retransmission. Duplicate acknowledgments are generated by a receiver when out-of-order segments arrive. As both segment loss and segment reordering cause out-of-order arrival, fast retransmit waits for three duplicate acknowledgments before considering the segment as lost. In some situations, however, the number of outstanding segments is not enough to trigger three duplicate acknowledgments, and the sender must rely on lengthy RTOs for loss recovery.

The amount of outstanding segments can be small for several reasons:

- (1) The connection is limited by the congestion control when the path has a low total capacity (bandwidth-delay product) or the connection's share of the capacity is small. It is also limited by the congestion control in the first RTTs of a connection or after an RTO when the available capacity is probed using slow-start.
- (2) The connection is limited by the receiver's available buffer space.
- (3) The connection is limited by the application if the available capacity of the path is not fully utilized (e.g. interactive applications), or at the end of a transfer, which is frequent if the total amount of data is small (e.g. web traffic).

The first two situations can occur for any flow, as external factors at the network and/or host level cause them. The third situation primarily affects flows that are short or have a low transmission rate. Typical examples of applications that produce short flows are web servers. [RJ10] shows that 70% of all web objects, found at the top 500 sites, are too small for fast retransmit to work. [BPS98]

shows that about 56% of all retransmissions sent by a busy web server are sent after RTO expiry. While the experiments were not conducted using SACK [RFC2018], only 4% of the RTO-based retransmissions could have been avoided. Applications have a low transmission rate when data is sent in response to actions, or as a reaction to real life events. Typical examples of such applications are stock trading systems, remote computer operations and online games. What is special about this class of applications is that they are time-dependant, and extra latency can reduce the application service level [P09]. Although such applications may represent a small amount of data sent on the network, a considerable number of flows have such properties and the importance of low latency is high.

The RTO restart approach outlined in this document makes the RTO slightly more aggressive when the number of outstanding segments is small, in an attempt to enable faster loss recovery for all segments while being robust to reordering. While it still conforms to the requirement in [RFC6298] that segments must not be retransmitted earlier than RTO seconds after their original transmission, it could increase the chance for a spurious timeout, which could degrade performance when the congestion window (cwnd) is large -- for example, when an application sends enough data to reach a cwnd covering 100 segments and then stops. The likelihood and potential impact of this problem as well as possible mitigation strategies are currently under investigation.

While this document focuses on TCP, the described changes are also valid for the Stream Control Transmission Protocol (SCTP) [RFC4960] which has similar loss recovery and congestion control algorithms.

### 1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## 2. RTO Restart Overview

The RTO management algorithm described in [RFC6298] recommends that the retransmission timer is restarted when an acknowledgment (ACK) that acknowledges new data is received and there is still outstanding data. The restart is conducted to guarantee that unacknowledged segments will be retransmitted after approximately RTO seconds. However, by restarting the timer on each incoming acknowledgment, retransmissions are not typically triggered RTO seconds after their previous transmission but rather RTO seconds after the last ACK arrived. The duration of this extra delay depends on several factors

but is in most cases approximately one RTT. Hence, in most situations the time before a retransmission is triggered is equal to "RTO + RTT".

The extra delay can be significant, especially for applications that use a lower RT<sub>min</sub> than the standard of 1 second and/or in environments with high RTTs, e.g. mobile networks. The restart approach is illustrated in Figure 1 where a TCP sender transmits three segments to a receiver. The arrival of the first and second segment triggers a delayed ACK [RFC1122], which restarts the RTO timer at the sender. The RTO restart is performed approximately one RTT after the transmission of the third segment. Thus, if the third segment is lost, as indicated in Figure 1, the effective loss detection time is "RTO + RTT" seconds. In some situations, the effective loss detection time becomes even longer. Consider a scenario where only two segments are outstanding. If the second segment is lost, the time to expire the delayed ACK timer will also be included in the effective loss detection time.

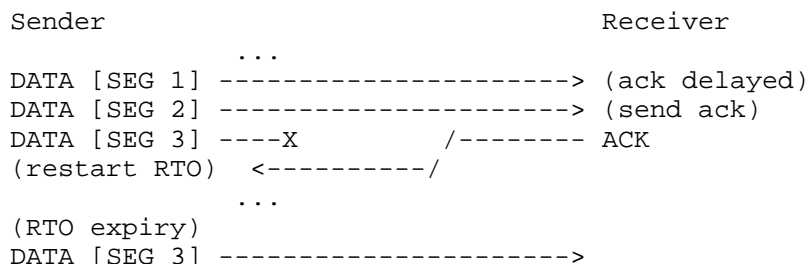


Figure 1: RTO restart example

During normal TCP bulk transfer the current RTO restart approach is not a problem. Actually, as long as enough segments arrive at a receiver to enable fast retransmit, RTO-based loss recovery should be avoided. RTOs should only be used as a last resort, as they drastically lower the congestion window compared to fast retransmit, and the current approach can therefore be beneficial -- it is described in [EL04] to act as a "safety margin" that compensates for some of the problems that the authors have identified with the standard RTO calculation. Notably, the authors of [EL04] also state that "this safety margin does not exist for highly interactive applications where often only a single packet is in flight."

There are only a few situations where timeouts are appropriate, or the only choice. For example, if the network is severely congested and no segments arrive, RTO-based recovery should be used. In this

situation, the time to recover from the loss(es) will not be the performance bottleneck. Furthermore, for connections that do not utilize enough capacity to enable fast retransmit, RTO is the only choice. The time needed for loss detection in such scenarios can become a serious performance bottleneck.

### 3. RTO Restart Algorithm

To enable faster loss recovery for connections that are unable to use fast retransmit, an alternative RTO restart can be used. By resetting the timer to "RTO - T\_earliest", where T\_earliest is the time elapsed since the earliest outstanding segment was transmitted, retransmissions will always occur after exactly RTO seconds. This approach makes the RTO more aggressive than the standardized approach in [RFC6298] but still conforms to the requirement in [RFC6298] that segments must not be retransmitted earlier than RTO seconds after their original transmission.

This document specifies the following update of step 5.3 in Section 5 of [RFC6298] (and a similar update in Section 6.3.2 of [RFC4960] for SCTP):

When an ACK is received that acknowledges new data:

- (1) Set T\_earliest = 0.
- (2) If the following two conditions hold:
  - (a) The number of outstanding segments is less than four.
  - (b) There is no unsent data ready for transmission or the receiver's advertised window does not permit transmission.set T\_earliest to the time elapsed since the earliest outstanding segment was sent.
- (3) Restart the retransmission timer so that it will expire after "RTO - T\_earliest" seconds (for the current value of RTO).

The update requires TCP implementations to track the time elapsed since the transmission of the earliest outstanding segment (T\_earliest). As the alternative restart is used only when the number of outstanding segments is less than four only four segments need to be tracked. Furthermore, some implementations of TCP (e.g. Linux TCP) already track the transmission times of all segments.

#### 4. Discussion

The currently standardized algorithm has been shown to add at least one RTT to the loss recovery process in TCP [LS00] and SCTP [HB08][PBP09]. Applications that have strict timing requirements (e.g. telephony signaling and gaming) rather than throughput requirements may want to use a lower RTT<sub>min</sub> than the standard of 1 second [RFC4166]. For such applications the modified restart approach could be important as the RTT and also the delayed ACK timer of receivers will be large components of the effective loss recovery time. Measurements in [HB08] have shown that the total transfer time of a lost segment (including the original transmission time and the loss recovery time) can be reduced with up to 35% using the suggested approach. These results match those presented in [PGH06][PBP09], where the modified restart approach is shown to significantly reduce retransmission latency.

There are several proposals that address the problem of not having enough ACKs for loss recovery. In what follows, we explain why the mechanism described here is complementary to these approaches:

The limited transmit mechanism [RFC3042] allows a TCP sender to transmit a previously unsent segment for each of the first two duplicate acknowledgments. By transmitting new segments, the sender attempts to generate additional duplicate acknowledgments to enable fast retransmit. However, limited transmit does not help if no previously unsent data is ready for transmission or if the receiver is out of buffer space. [RFC5827] specifies an early retransmit algorithm to enable fast loss recovery in such situations. By dynamically lowering the amount of duplicate acknowledgments needed for fast retransmit (dupthresh), based on the number of outstanding segments, a smaller number of duplicate acknowledgments are needed to trigger a retransmission. In some situations, however, the algorithm is of no use or might not work properly. First, if a single segment is outstanding, and lost, it is impossible to use early retransmit. Second, if ACKs are lost, the early retransmit cannot help. Third, if the network path reorders segments, the algorithm might cause more unnecessary retransmissions than fast retransmit.

TCP-NCR [RFC4653] sets the dupthresh to three or more, to better disambiguate reordered and lost segments. In addition, early retransmit lowers the dupthresh when the amount of outstanding data is small, to enable faster loss recovery. The reasons why the RTO restart procedure described in this document does not take dynamic dupthresh considerations into account are twofold. First, if a larger dupthresh is used, the RTO restart approach could be used when the congestion window, and the amount of outstanding data, is larger. However, in such situations the actual amount of outstanding data can

significantly impact the RTT of the connection, making it potentially dangerous to be more aggressive. Second, if a smaller dupthresh is used, the amount of outstanding data needed for a restart is smaller. However, as the congestion window is already small, it does not matter if a retransmission is due to a fast retransmit or an RTO. The resulting congestion window will still be very small, and the only difference is how quickly TCP infers segment loss.

Tail Loss Probe [TLP] is a proposal to send up to two "probe segments" when a timer fires which is set to a value smaller than the RTO. A "probe segment" is a new segment if new data is available, else a retransmission. The intention is to compensate for sluggish RTO behavior in situations where the RTO greatly exceeds the RTT, which, according to measurements reported in [TLP], is not uncommon. The Probe timeout (PTO) is at least 2 RTTs, and only scheduled in case the RTO is farther than the PTO. A spurious PTO is less risky than a spurious RTO, as it would not have the same negative effects (clearing the scoreboard and restarting with slow-start). In contrast, RTO restart is trying to make the RTO more appropriate in cases where there is no need to be overly cautious.

TLP could kick in in situations where RTO restart does not apply, and it could overrule (yielding a similar general behavior, but with a lower timeout) RTO restart in cases where the number of outstanding segments is smaller than 4 and no new segments are available for transmission. The shorter RTO from RTO restart also reduces the probability that TLP is activated because PTO might be farther than RTO. This could make RTO restart more aggressive than the algorithm in [TLP] when:

- (1) no data has been sent in an interval exceeding the RTO
- (2) the number of outstanding segments is 3
- (3) (defined in [RFC5681]) is at least 3

because, under these conditions, in accordance with [RFC5681], 3 packets can immediately be retransmitted, whereas TLP only allows up to two consecutive PTOs.

## 5. IANA Considerations

This memo includes no request to IANA.

## 6. Security Considerations

This document discusses a change in how to set the retransmission timer's value when restarted. This change does not raise any new security issues with TCP or SCTP.

## 7. References

### 7.1. Normative References

- [RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.
- [RFC4166] Coene, L. and J. Pastor-Balbas, "Telephony Signalling Transport over Stream Control Transmission Protocol (SCTP) Applicability Statement", RFC 4166, February 2006.
- [RFC4653] Bhandarkar, S., Reddy, A., Allman, M., and E. Blanton, "Improving the Robustness of TCP to Non-Congestion Events", RFC 4653, August 2006.
- [RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC5827] Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J., and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)", RFC 5827, May 2010.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

## 7.2. Informative References

- [BPS98] Balakrishnan, H., Padmanabhan, V., Seshan, S., Stemm, M., and R. Katz, "TCP Behavior of a Busy Web Server: Analysis and Improvements", Proc. IEEE INFOCOM Conf., March 1998.
- [EL04] Ekstroem, H. and R. Ludwig, "The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport", IEEE INFOCOM 2004, March 2004.
- [HB08] Hurtig, P. and A. Brunstrom, "SCTP: designed for timely message delivery?", Springer Telecommunication Systems, May 2010.
- [LS00] Ludwig, R. and K. Sklower, "The Eifel retransmission timer", ACM SIGCOMM Comput. Commun. Rev., 30(3), July 2000.
- [P09] Petlund, A., "Improving latency for interactive, thin-stream applications over reliable transport", Unipub PhD Thesis, Oct 2009.
- [PBP09] Petlund, A., Beskow, P., Pedersen, J., Paaby, E., Griwodz, C., and P. Halvorsen, "Improving SCTP Retransmission Delays for Time-Dependent Thin Streams", Springer Multimedia Tools and Applications, 45(1-3), 2009.
- [PGH06] Pedersen, J., Griwodz, C., and P. Halvorsen, "Considerations of SCTP Retransmission Delays for Thin Streams", IEEE LCN 2006, November 2006.
- [RJ10] Ramachandran, S., "Web metrics: Size and number of resources", Google <http://code.google.com/speed/articles/web-metrics.html>, May 2010.
- [TLP] Dukkipati, N., Cardwell, N., Cheng, Y., and M. Mathis, "TCP Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses", draft-dukkipati-tcpm-tcp-loss-probe-00.txt (work in progress), July 2012.

Authors' Addresses

Per Hurtig  
Karlstad University  
Universitetsgatan 2  
Karlstad, 651 88  
Sweden

Phone: +46 54 700 23 35  
Email: per.hurtig@kau.se

Andreas Petlund  
Simula Research Laboratory AS  
P.O. Box 134  
Lysaker, 1325  
Norway

Phone: +47 67 82 82 00  
Email: apetlund@simula.no

Michael Welzl  
University of Oslo  
PO Box 1080 Blindern  
Oslo, N-0316  
Norway

Phone: +47 22 85 24 20  
Email: michawe@ifi.uio.no



TCP Maintenance (TCPM)  
Internet-Draft  
Obsoletes: 1323 (if approved)  
Intended status: Standards Track  
Expires: October 13, 2014

D. Borman  
Quantum Corporation  
B. Braden  
University of Southern  
California  
V. Jacobson  
Google, Inc.  
R. Scheffenegger, Ed.  
NetApp, Inc.  
April 11, 2014

TCP Extensions for High Performance  
draft-ietf-tcpm-1323bis-21

Abstract

This document specifies a set of TCP extensions to improve performance over paths with a large bandwidth \* delay product and to provide reliable operation over very high-speed paths. It defines the TCP Window Scale (WS) option and the TCP Timestamps (TS) option and their semantics. The Window Scale option is used to support larger receive windows, while the Timestamps option can be used for at least two distinct mechanisms, PAWS (Protection Against Wrapped Sequences) and RTTM (Round Trip Time Measurement), that are also described herein.

This document obsoletes RFC1323 and describes changes from it.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 13, 2014.

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	4
1.1. TCP Performance . . . . .	4
1.2. TCP Reliability . . . . .	5
1.3. Using TCP options . . . . .	6
1.4. Terminology . . . . .	7
2. TCP Window Scale option . . . . .	8
2.1. Introduction . . . . .	8
2.2. Window Scale option . . . . .	8
2.3. Using the Window Scale option . . . . .	9
2.4. Addressing Window Retraction . . . . .	10
3. TCP Timestamps option . . . . .	12
3.1. Introduction . . . . .	12
3.2. Timestamps option . . . . .	12
4. The RTTM Mechanism . . . . .	15
4.1. Introduction . . . . .	15
4.2. Updating the RTO value . . . . .	16
4.3. Which Timestamp to Echo . . . . .	16
5. PAWS - Protection Against Wrapped Sequence Numbers . . . . .	20
5.1. Introduction . . . . .	20
5.2. The PAWS Mechanism . . . . .	20
5.3. Basic PAWS Algorithm . . . . .	21
5.4. Timestamp Clock . . . . .	23
5.5. Outdated Timestamps . . . . .	25
5.6. Header Prediction . . . . .	25
5.7. IP Fragmentation . . . . .	27
5.8. Duplicates from Earlier Incarnations of Connection . . . . .	27
6. Conclusions and Acknowledgments . . . . .	28
7. Security Considerations . . . . .	28
7.1. Privacy Considerations . . . . .	30
8. IANA Considerations . . . . .	30
9. References . . . . .	30
9.1. Normative References . . . . .	30
9.2. Informative References . . . . .	31
Appendix A. Implementation Suggestions . . . . .	34
Appendix B. Duplicates from Earlier Connection Incarnations . . . . .	35
B.1. System Crash with Loss of State . . . . .	35
B.2. Closing and Reopening a Connection . . . . .	36
Appendix C. Summary of Notation . . . . .	37
Appendix D. Event Processing Summary . . . . .	38
Appendix E. Timestamps Edge Cases . . . . .	43
Appendix F. Window Retraction Example . . . . .	44
Appendix G. RTO calculation modification . . . . .	45
Appendix H. Changes from RFC 1323 . . . . .	45
Authors' Addresses . . . . .	48

## 1. Introduction

The TCP protocol [RFC0793] was designed to operate reliably over almost any transmission medium regardless of transmission rate, delay, corruption, duplication, or reordering of segments. Over the years, advances in networking technology have resulted in ever-higher transmission speeds, and the fastest paths are well beyond the domain for which TCP was originally engineered.

This document defines a set of modest extensions to TCP to extend the domain of its application to match the increasing network capability. It is an update to and obsoletes [RFC1323], which in turn is based upon and obsoletes [RFC1072] and [RFC1185].

Changes between [RFC1323] and this document are detailed in Appendix H. These changes are partly due to errata in [RFC1323], and partly due to the improved understanding of how the involved components interact.

For brevity, the full discussions of the merits and history behind the TCP options defined within this document have been omitted. [RFC1323] should be consulted for reference. It is recommended that a modern TCP stack implements and make use of the extensions described in this document.

### 1.1. TCP Performance

TCP performance problems arise when the bandwidth \* delay product is large. A network having such paths is referred to as "long, fat network" (LFN).

There are two fundamental performance problems with basic TCP over LFN paths:

#### (1) Window Size Limit

The TCP header uses a 16 bit field to report the receive window size to the sender. Therefore, the largest window that can be used is  $2^{16} = 64$  KiB. For LFN paths where the bandwidth \* delay product exceeds 64 KiB, the receive window limits the maximum throughput of the TCP connection over the path, i.e., the amount of unacknowledged data that TCP can send in order to keep the pipeline full.

To circumvent this problem, Section 2 of this memo defines a TCP option, "Window Scale", to allow windows larger than  $2^{16}$ . This option defines an implicit scale factor, which is used to multiply the window size value found in a TCP header to obtain

the true window size.

It must be noted, that the use of large receive windows increases the chance of too quickly wrapping sequence numbers, as described below in Section 1.2, (1).

## (2) Recovery from Losses

Packet losses in an LFN can have a catastrophic effect on throughput.

To generalize the Fast Retransmit / Fast Recovery mechanism to handle multiple packets dropped per window, Selective Acknowledgments are required. Unlike the normal cumulative acknowledgments of TCP, Selective Acknowledgments give the sender a complete picture of which segments are queued at the receiver and which have not yet arrived.

Selective acknowledgments and their use are specified in separate documents, "TCP Selective Acknowledgment options" [RFC2018], "An Extension to the Selective Acknowledgement (SACK) option for TCP" [RFC2883], and "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP" [RFC6675], and not further discussed in this document.

## 1.2. TCP Reliability

An especially serious kind of error may result from an accidental reuse of TCP sequence numbers in data segments. TCP reliability depends upon the existence of a bound on the lifetime of a segment: the "Maximum Segment Lifetime" or MSL.

Duplication of sequence numbers might happen in either of two ways:

### (1) Sequence number wrap-around on the current connection

A TCP sequence number contains 32 bits. At a high enough transfer rate of large volumes of data (at least 4 GiB in the same session), the 32-bit sequence space may be "wrapped" (cycled) within the time that a segment is delayed in queues.

### (2) Earlier incarnation of the connection

Suppose that a connection terminates, either by a proper close sequence or due to a host crash, and the same connection (i.e., using the same pair of port numbers) is immediately reopened. A delayed segment from the terminated connection could fall within the current window for the new incarnation and be accepted as

valid.

Duplicates from earlier incarnations, case (2), are avoided by enforcing the current fixed MSL of the TCP specification, as explained in Section 5.8 and Appendix B. In addition, the randomizing of ephemeral ports can also help to probabilistically reduce the chances of duplicates from earlier connections. However, case (1), avoiding the reuse of sequence numbers within the same connection, requires an upper bound on MSL that depends upon the transfer rate, and at high enough rates, a dedicated mechanism is required.

A possible fix for the problem of cycling the sequence space would be to increase the size of the TCP sequence number field. For example, the sequence number field (and also the acknowledgment field) could be expanded to 64 bits. This could be done either by changing the TCP header or by means of an additional option.

Section 5 presents a different mechanism, which we call PAWS (Protection Against Wrapped Sequence numbers), to extend TCP reliability to transfer rates well beyond the foreseeable upper limit of network bandwidths. PAWS uses the TCP Timestamps option defined in Section 3.2 to protect against old duplicates from the same connection.

### 1.3. Using TCP options

The extensions defined in this document all use TCP options.

When [RFC1323] was published, there was concern that some buggy TCP implementation might crash on the first appearance of an option on a non-`<SYN>` segment. However, bugs like that can lead to DOS attacks against a TCP. Research has shown that most TCP implementations will properly handle unknown options on non-`<SYN>` segments ([Medina04], [Medina05]). But it is still prudent to be conservative in what you send, and avoiding buggy TCP implementation is not the only reason for negotiating TCP options on `<SYN>` segments.

The window scale option negotiates fundamental parameters of the TCP session. Therefore, it is only sent during the initial handshake. Furthermore, the window scale option will be sent in a `<SYN,ACK>` segment only if the corresponding option was received in the initial `<SYN>` segment.

The Timestamps option may appear in any data or `<ACK>` segment, adding 10 bytes (up to 12 bytes including padding) to the 20-byte TCP header. It is required that this TCP option will be sent on all non-`<SYN>` segments after an exchange of options on the `<SYN>` segments has

indicated that both sides understand this extension.

Research has shown that the use of the Timestamps option to take additional RTT samples within each RTT has little effect on the ultimate retransmission timeout value [Allman99]. However, there are other uses of the Timestamps option, such as the Eifel mechanism [RFC3522], [RFC4015], and PAWS (see Section 5) which improve overall TCP security and performance. The extra header bandwidth used by this option should be evaluated for the gains in performance and security in an actual deployment.

Appendix A contains a recommended layout of the options in TCP headers to achieve reasonable data field alignment.

Finally, we observe that most of the mechanisms defined in this document are important for LFNs and/or very high-speed networks. For low-speed networks, it might be a performance optimization to NOT use these mechanisms. A TCP vendor concerned about optimal performance over low-speed paths might consider turning these extensions off for low-speed paths, or allow a user or installation manager to disable them.

#### 1.4. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

In this document, these words will appear with that interpretation only when in UPPER CASE. Lower case uses of these words are not to be interpreted as carrying [RFC2119] significance.

## 2. TCP Window Scale option

### 2.1. Introduction

The window scale extension expands the definition of the TCP window to 30 bits and then uses an implicit scale factor to carry this 30-bit value in the 16-bit Window field of the TCP header (SEG.WND in [RFC0793]). The exponent of the scale factor is carried in a TCP option, Window Scale. This option is sent only in a <SYN> segment (a segment with the SYN bit on), hence the window scale is fixed in each direction when a connection is opened.

The maximum receive window, and therefore the scale factor, is determined by the maximum receive buffer space. In a typical modern implementation, this maximum buffer space is set by default but can be overridden by a user program before a TCP connection is opened. This determines the scale factor, and therefore no new user interface is needed for window scaling.

### 2.2. Window Scale option

The three-byte Window Scale option MAY be sent in a <SYN> segment by a TCP. It has two purposes: (1) indicate that the TCP is prepared to both send and receive window scaling, and (2) communicate the exponent of a scale factor to be applied to its receive window. Thus, a TCP that is prepared to scale windows SHOULD send the option, even if its own scale factor is 1 and the exponent 0. The scale factor is limited to a power of two and encoded logarithmically, so it may be implemented by binary shift operations. The maximum scale exponent is limited to 14 for a maximum permissible receive window size of 1 GiB ( $2^{(14+16)}$ ).

TCP Window Scale option (WSopt):

Kind: 3

Length: 3 bytes

```

+-----+-----+-----+
| Kind=3 | Length=3 | shift.cnt |
+-----+-----+-----+
      1         1         1

```

This option is an offer, not a promise; both sides MUST send Window Scale options in their <SYN> segments to enable window scaling in either direction. If window scaling is enabled, then the TCP that sent this option will right-shift its true receive-window values by 'shift.cnt' bits for transmission in SEG.WND. The value 'shift.cnt'

MAY be zero (offering to scale, while applying a scale factor of 1 to the receive window).

This option MAY be sent in an initial <SYN> segment (i.e., a segment with the SYN bit on and the ACK bit off). If a Window Scale option was received in the initial <SYN> segment, then this option MAY be sent in the <SYN,ACK> segment. A Window Scale option in a segment without a SYN bit MUST be ignored.

The window field in a segment where the SYN bit is set (i.e., a <SYN> or <SYN,ACK>) MUST NOT be scaled.

### 2.3. Using the Window Scale option

A model implementation of window scaling is as follows, using the notation of [RFC0793]:

- o The connection state is augmented by two window shift counters, Snd.Wind.Shift and Rcv.Wind.Shift, to be applied to the incoming and outgoing window fields, respectively.
- o If a TCP receives a <SYN> segment containing a Window Scale option, it SHOULD send its own Window Scale option in the <SYN,ACK> segment.
- o The Window Scale option MUST be sent with shift.cnt = R, where R is the value that the TCP would like to use for its receive window.
- o Upon receiving a <SYN> segment with a Window Scale option containing shift.cnt = S, a TCP MUST set Snd.Wind.Shift to S and MUST set Rcv.Wind.Shift to R; otherwise, it MUST set both Snd.Wind.Shift and Rcv.Wind.Shift to zero.
- o The window field (SEG.WND) in the header of every incoming segment, with the exception of <SYN> segments, MUST be left-shifted by Snd.Wind.Shift bits before updating SND.WND:

$$\text{SND.WND} = \text{SEG.WND} \ll \text{Snd.Wind.Shift}$$

(assuming the other conditions of [RFC0793] are met, and using the "C" notation "<<" for left-shift).

- o The window field (SEG.WND) of every outgoing segment, with the exception of <SYN> segments, MUST be right-shifted by Rcv.Wind.Shift bits:

$$\text{SEG.WND} = \text{RCV.WND} \gg \text{Rcv.Wind.Shift}$$

TCP determines if a data segment is "old" or "new" by testing whether its sequence number is within  $2^{31}$  bytes of the left edge of the window, and if it is not, discarding the data as "old". To insure that new data is never mistakenly considered old and vice versa, the left edge of the sender's window has to be at most  $2^{31}$  away from the right edge of the receiver's window. Similarly with the sender's right edge and receiver's left edge. Since the right and left edges of either the sender's or receiver's window differ by the window size, and since the sender and receiver windows can be out of phase by at most the window size, the above constraints imply that two times the maximum window size must be less than  $2^{31}$ , or

$$\text{max window} < 2^{30}$$

Since the max window is  $2^S$  (where  $S$  is the scaling shift count) times at most  $2^{16} - 1$  (the maximum unscaled window), the maximum window is guaranteed to be  $< 2^{30}$  if  $S \leq 14$ . Thus, the shift count MUST be limited to 14 (which allows windows of  $2^{30} = 1 \text{ GiB}$ ). If a Window Scale option is received with a shift.cnt value larger than 14, the TCP SHOULD log the error but MUST use 14 instead of the specified value. This is safe as a sender can always choose to only partially use any signaled receive window. If the receiver is scaling by a factor larger than 14 and the sender is only scaling by 14 then the receive window used by the sender will appear smaller than it is in reality.

The scale factor applies only to the Window field as transmitted in the TCP header; each TCP using extended windows will maintain the window values locally as 32-bit numbers. For example, the "congestion window" computed by Slow Start and Congestion Avoidance (see [RFC5681]) is not affected by the scale factor, so window scaling will not introduce quantization into the congestion window.

#### 2.4. Addressing Window Retraction

When a non-zero scale factor is in use, there are instances when a retracted window can be offered - see Appendix F for a detailed example. The end of the window will be on a boundary based on the granularity of the scale factor being used. If the sequence number is then updated by a number of bytes smaller than that granularity, the TCP will have to either advertise a new window that is beyond what it previously advertised (and perhaps beyond the buffer), or will have to advertise a smaller window, which will cause the TCP window to shrink. Implementations MUST ensure that they handle a shrinking window, as specified in section 4.2.2.16 of [RFC1122].

For the receiver, this implies that:

- 1) The receiver MUST honor, as in-window, any segment that would have been in-window for any <ACK> sent by the receiver.
- 2) When window scaling is in effect, the receiver SHOULD track the actual maximum window sequence number (which is likely to be greater than the window announced by the most recent <ACK>, if more than one segment has arrived since the application consumed any data in the receive buffer).

On the sender side:

- 3) The initial transmission MUST be within the window announced by the most recent <ACK>.
- 4) On first retransmission, or if the sequence number is out-of-window by less than  $2^{\text{Rcv.Wind.Shift}}$  then do normal retransmission(s) without regard to receiver window as long as the original segment was in window when it was sent.
- 5) Subsequent retransmissions MAY only be sent, if they are within the window announced by the most recent <ACK>.

### 3. TCP Timestamps option

#### 3.1. Introduction

The Timestamps option is introduced to address some of the issues mentioned in Section 1.1 and Section 1.2. The Timestamps option is specified in a symmetrical manner, so that TSval timestamps are carried in both data and <ACK> segments and are echoed in TSecr fields carried in returning <ACK> or data segments. Originally used primarily for timestamping individual segments, the properties of the Timestamps option allow not only the use for taking time measurements (Section 4), but additional uses as well (Section 5).

It is necessary to remember that there is a distinction between the Timestamps option conveying timestamp information, and the use of that information. In particular, the Round Trip Time Measurement (RTTM) mechanism must be viewed independently from updating the Retransmission Timeout (RTO) (see Section 4.2). In this case, the sample granularity also needs to be taken into account. Other mechanisms, such as PAWS, or Eifel, are not built upon the timestamp information itself, but are based on the intrinsic property of monotonically non-decreasing values.

The Timestamps option is important when large receive windows are used, to allow the use of the PAWS mechanism (see Section 5). Furthermore, the option may be useful for all TCPs, since it simplifies the sender and allows the use of additional optimizations such as Eifel ([RFC3522], [RFC4015]) and others ([RFC6817], [Kuzmanovic03], [Kuehlewind10]).

#### 3.2. Timestamps option

TCP is a symmetric protocol, allowing data to be sent at any time in either direction, and therefore timestamp echoing may occur in either direction. For simplicity and symmetry, we specify that timestamps always be sent and echoed in both directions. For efficiency, we combine the timestamp and timestamp reply fields into a single TCP Timestamps option.

TCP Timestamps option (TSopt):

Kind: 8

Length: 10 bytes

Kind=8	10	TS Value (TSval)	TS Echo Reply (TSecr)
1	1	4	4

The Timestamps option carries two four-byte timestamp fields. The Timestamp Value field (TSval) contains the current value of the timestamp clock of the TCP sending the option.

The Timestamp Echo Reply (TSecr) field is valid if the ACK bit is set in the TCP header. If the ACK bit is not set in the outgoing TCP header, the sender of that segment SHOULD set the TSecr field to zero. When the ACK bit is set in an outgoing segment, the sender MUST echo a recently received Timestamp Value (TSval) sent by the remote TCP in the TSval field of a Timestamps option. The exact rules on which TSval MUST be echoed are given in Section 4.3. When the ACK bit is not set, the receiver MUST ignore the value of the TSecr field.

A TCP MAY send the Timestamps option (TSopt) in an initial <SYN> segment (i.e., segment containing a SYN bit and no ACK bit), and MAY send a TSopt in <SYN,ACK> only if it received a TSopt in the initial <SYN> segment for the connection.

Once TSopt has been successfully negotiated, that is both <SYN>, and <SYN,ACK> contain TSopt, the TSopt MUST be sent in every non-<RST> segment for the duration of the connection, and SHOULD be sent in an <RST> segment (see Section 5.2 for details). The TCP SHOULD remember this state by setting a flag, referred to as Snd.TS.OK, to one. If a non-<RST> segment is received without a TSopt, a TCP SHOULD silently drop the segment. A TCP MUST NOT abort a TCP connection because any segment lacks an expected TSopt.

Implementations are strongly encouraged to follow the above rules for handling a missing Timestamps option, and the order of precedence mentioned in Section 5.3 when deciding on the acceptance of a segment.

If a receiver chooses to accept a segment without an expected Timestamps option, it must be clear that undetectable data corruption may occur.

Such a TCP receiver may experience undetectable wrapped- sequence effects, such as data (payload) corruption or session stalls. In order to maintain the integrity of the payload data, in particular on high speed networks, it is paramount to follow the described processing rules.

However, it has been mentioned that under some circumstances, the above guidelines are too strict, and some paths sporadically suppress the Timestamps option, while maintaining payload integrity. A path behaving in this manner should be deemed unacceptable, but it has been noted that some implementations relax the acceptance rules as a workaround, and allow TCP to run across such paths [Oppermann13]

If a TSopt is received on a connection where TSopt was not negotiated in the initial three-way handshake, the TSopt MUST be ignored and the packet processed normally.

In the case of crossing <SYN> segments where one <SYN> contains a TSopt and the other doesn't, both sides MAY send a TSopt in the <SYN,ACK> segment.

TSopt is required for the two mechanisms described in sections 4 and 5. There are also other mechanisms that rely on the presence of the TSopt, e.g. [RFC3522]. If a TCP stopped sending TSopt at any time during an established session, it interferes with these mechanisms. This update to [RFC1323] describes explicitly the previous assumption (see Section 5.2), that each TCP segment must have TSopt, once negotiated.

## 4. The RTTM Mechanism

### 4.1. Introduction

One use of the Timestamps option is to measure the round trip time of virtually every packet acknowledged. The Round Trip Time Measurement (RTTM) mechanism requires a Timestamps option in every measured segment, with a TSval that is obtained from a (virtual) "timestamp clock". Values of this clock MUST be at least approximately proportional to real time, in order to measure actual RTT.

TCP measures the round trip time (RTT), primarily for the purpose of arriving at a reasonable value for the Retransmission Timeout (RTO) timer interval. Accurate and current RTT estimates are necessary to adapt to changing traffic conditions, while a conservative estimate of the RTO interval is necessary to minimize spurious RTOs.

These TSval values are echoed in TSecr values in the reverse direction. The difference between a received TSecr value and the current timestamp clock value provides an RTT measurement.

When timestamps are used, every segment that is received will contain a TSecr value. However, these values cannot all be used to update the measured RTT. The following example illustrates why. It shows a one-way data flow with segments arriving in sequence without loss. Here A, B, C... represent data blocks occupying successive blocks of sequence numbers, and ACK(A),... represent the corresponding cumulative acknowledgments. The two timestamp fields of the Timestamps option are shown symbolically as <TSval=x,TSecr=y>. Each TSecr field contains the value most recently received in a TSval field.

```

TCP A                                     TCP B

                                <A,TSval=1,TSecr=120> ----->

<----- <ACK(A),TSval=127,TSecr=1>

                                <B,TSval=5,TSecr=127> ----->

<----- <ACK(B),TSval=131,TSecr=5>

. . . . .

                                <C,TSval=65,TSecr=131> ----->

<----- <ACK(C),TSval=191,TSecr=65>

```

(etc.)

The dotted line marks a pause (60 time units long) in which A had nothing to send. Note that this pause inflates the RTT which B could infer from receiving TSecr=131 in data segment C. Thus, in one-way data flows, RTTM in the reverse direction measures a value that is inflated by gaps in sending data. However, the following rule prevents a resulting inflation of the measured RTT:

RTTM Rule: A TSecr value received in a segment MAY be used to update the averaged RTT measurement only if the segment advances the left edge of the send window, i.e. SND.UNA is increased.

Since TCP B is not sending data, the data segment C does not acknowledge any new data when it arrives at B. Thus, the inflated RTTM measurement is not used to update B's RTTM measurement.

#### 4.2. Updating the RTO value

When [RFC1323] was originally written, it was perceived that taking RTT measurements for each segment, and also during retransmissions, would contribute to reduce spurious RTOs, while maintaining the timeliness of necessary RTOs. At the time, RTO was also the only mechanism to make use of the measured RTT. It has been shown, that taking more RTT samples has only a very limited effect to optimize RTOs [Allman99].

Implementers should note that with timestamps multiple RTTMs can be taken per RTT. The [RFC6298] RTO estimator has weighting factors, alpha and beta, based on an implicit assumption that at most one RTTM will be sampled per RTT. When multiple RTTMs per RTT are available to update the RTO estimator, an implementation SHOULD try to adhere to the spirit of the history specified in [RFC6298]. An implementation suggestion is detailed in Appendix G.

[Ludwig00] and [Floyd05] have highlighted the problem that an unmodified RTO calculation, which is updated with per-packet RTT samples, will truncate the path history too soon. This can lead to an increase in spurious retransmissions, when the path properties vary in the order of a few RTTs, but a high number of RTT samples are taken on a much shorter timescale.

#### 4.3. Which Timestamp to Echo

If more than one Timestamps option is received before a reply segment is sent, the TCP must choose only one of the TSvals to echo, ignoring the others. To minimize the state kept in the receiver (i.e., the

number of unprocessed TSvals), the receiver should be required to retain at most one timestamp in the connection control block.

There are three situations to consider:

(A) Delayed ACKs.

Many TCPs acknowledge only every second segment out of a group of segments arriving within a short time interval; this policy is known generally as "delayed ACKs". The data-sender TCP must measure the effective RTT, including the additional time due to delayed ACKs, or else it will retransmit unnecessarily. Thus, when delayed ACKs are in use, the receiver SHOULD reply with the TSval field from the earliest unacknowledged segment.

(B) A hole in the sequence space (segment(s) have been lost).

The sender will continue sending until the window is filled, and the receiver may be generating <ACK>s as these out-of-order segments arrive (e.g., to aid "fast retransmit").

The lost segment is probably a sign of congestion, and in that situation the sender should be conservative about retransmission. Furthermore, it is better to overestimate than underestimate the RTT. An <ACK> for an out-of-order segment SHOULD therefore contain the timestamp from the most recent segment that advanced RCV.NXT.

The same situation occurs if segments are re-ordered by the network.

(C) A filled hole in the sequence space.

The segment that fills the hole and advances the window represents the most recent measurement of the network characteristics. An RTT computed from an earlier segment would probably include the sender's retransmit time-out, badly biasing the sender's average RTT estimate. Thus, the timestamp from the latest segment (which filled the hole) MUST be echoed.

An algorithm that covers all three cases is described in the following rules for Timestamps option processing on a synchronized connection:

(1) The connection state is augmented with two 32-bit slots:

TS.Recent holds a timestamp to be echoed in TSecr whenever a segment is sent, and Last.ACK.sent holds the ACK field from the

last segment sent. Last.ACK.sent will equal RCV.NXT except when <ACK>s have been delayed.

(2) If:

SEG.TSval >= TS.recent and SEG.SEQ <= Last.ACK.sent

then SEG.TSval is copied to TS.Recent; otherwise, it is ignored.

(3) When a TSopt is sent, its TSecr field is set to the current TS.Recent value.

The following examples illustrate these rules. Here A, B, C... represent data segments occupying successive blocks of sequence numbers, and ACK(A),... represent the corresponding acknowledgment segments. Note that ACK(A) has the same sequence number as B. We show only one direction of timestamp echoing, for clarity.

o Segments arrive in sequence, and some of the <ACK>s are delayed.

By case (A), the timestamp from the oldest unacknowledged segment is echoed.

	TS.Recent
<A, TSval=1> ----->	1
<B, TSval=2> ----->	1
<C, TSval=3> ----->	1
<---- <ACK(C), TSecr=1>	
(etc)	

o Segments arrive out of order, and every segment is acknowledged.

By case (B), the timestamp from the last segment that advanced the left window edge is echoed, until the missing segment arrives; it is echoed according to Case (C). The same sequence would occur if segments B and D were lost and retransmitted.

	TS.Recent
<A, TSval=1> ----->	1
<----- <ACK(A), TSecr=1>	1
<C, TSval=3> ----->	1
<----- <ACK(A), TSecr=1>	1
<B, TSval=2> ----->	2
<----- <ACK(C), TSecr=2>	2
<E, TSval=5> ----->	2
<----- <ACK(C), TSecr=2>	2
<D, TSval=4> ----->	4
<----- <ACK(E), TSecr=4>	
(etc)	

## 5. PAWS - Protection Against Wrapped Sequence Numbers

### 5.1. Introduction

Another use for the Timestamps options is the mechanism to Protect Against Wrapped Sequence numbers (PAWS). Section 5.2 describes a simple mechanism to reject old duplicate segments that might corrupt an open TCP connection. PAWS operates within a single TCP connection, using state that is saved in the connection control block. Section 5.8 and Appendix H discuss the implications of the PAWS mechanism for avoiding old duplicates from previous incarnations of the same connection.

### 5.2. The PAWS Mechanism

PAWS uses the TCP Timestamps option described earlier, and assumes that every received TCP segment (including data and <ACK> segments) contains a timestamp `SEG.TSval` whose values are monotonically non-decreasing in time. The basic idea is that a segment can be discarded as an old duplicate if it is received with a timestamp `SEG.TSval` less than some timestamp recently received on this connection.

In the PAWS mechanism, the "timestamps" are 32-bit unsigned integers in a modular 32-bit space. Thus, "less than" is defined the same way it is for TCP sequence numbers, and the same implementation techniques apply. If  $s$  and  $t$  are timestamp values,

$$s < t \text{ if } 0 < (t - s) < 2^{31},$$

computed in unsigned 32-bit arithmetic.

The choice of incoming timestamps to be saved for this comparison MUST guarantee a value that is monotonically non-decreasing. For example, an implementation might save the timestamp from the segment that last advanced the left edge of the receive window, i.e., the most recent in-sequence segment. For simplicity, the value `TS.Recent` introduced in Section 4.3 is used instead, as using a common value for both PAWS and RTTM simplifies the implementation. As Section 4.3 explained, `TS.Recent` differs from the timestamp from the last in-sequence segment only in the case of delayed <ACK>s, and therefore by less than one window. Either choice will therefore protect against sequence number wrap-around.

PAWS submits all incoming segments to the same test, and therefore protects against duplicate <ACK> segments as well as data segments. (An alternative non-symmetric algorithm would protect against old duplicate <ACK>s: the sender of data would reject incoming <ACK>

segments whose TSecr values were less than the TSecr saved from the last segment whose ACK field advanced the left edge of the send window. This algorithm was deemed to lack economy of mechanism and symmetry.)

TSval timestamps sent on <SYN> and <SYN,ACK> segments are used to initialize PAWS. PAWS protects against old duplicate non- <SYN> segments, and duplicate <SYN> segments received while there is a synchronized connection. Duplicate <SYN> and <SYN,ACK> segments received when there is no connection will be discarded by the normal 3-way handshake and sequence number checks of TCP.

[RFC1323] recommended that <RST> segments NOT carry timestamps, and that they be acceptable regardless of their timestamp. At that time, the thinking was that old duplicate <RST> segments should be exceedingly unlikely, and their cleanup function should take precedence over timestamps. More recently, discussions about various blind attacks on TCP connections have raised the suggestion that if the Timestamps option is present, SEG.TSecr could be used to provide stricter acceptance tests for <RST> segments.

While still under discussion, to enable research into this area it is now RECOMMENDED that when generating an <RST>, that if the segment causing the <RST> to be generated contained a Timestamps option, that the <RST> also contain a Timestamps option. In the <RST> segment, SEG.TSecr SHOULD be set to SEG.TSval from the incoming segment and SEG.TSval SHOULD be set to zero. If an <RST> is being generated because of a user abort, and Snd.TS.OK is set, then a Timestamps option SHOULD be included in the <RST>. When an <RST> segment is received, it MUST NOT be subjected to the PAWS check by verifying an acceptable value in SEG.TSval, and information from the Timestamps option MUST NOT be used to update connection state information. SEG.TSecr MAY be used to provide stricter <RST> acceptance checks.

### 5.3. Basic PAWS Algorithm

If the PAWS algorithm is used, the following processing MUST be performed on all incoming segments for a synchronized connection. Also, PAWS processing MUST take precedence over the regular TCP acceptability check (Section 3.3 in [RFC0793]), which is performed after verification of the received Timestamps option:

- R1) If there is a Timestamps option in the arriving segment, SEG.TSval < TS.Recent, TS.Recent is valid (see later discussion) and the RST bit is not set, then treat the arriving segment as not acceptable:

Send an acknowledgment in reply as specified in [RFC0793] page 69 and drop the segment.

Note: it is necessary to send an <ACK> segment in order to retain TCP's mechanisms for detecting and recovering from half-open connections. For example, see Figure 10 of [RFC0793].

- R2) If the segment is outside the window, reject it (normal TCP processing)
- R3) If an arriving segment satisfies: `SEG.SEQ <= Last.ACK.sent` (see Section 4.3), then record its timestamp in `TS.Recent`.
- R4) If an arriving segment is in-sequence (i.e., at the left window edge), then accept it normally.
- R5) Otherwise, treat the segment as a normal in-window, out-of-sequence TCP segment (e.g., queue it for later delivery to the user).

Steps R2, R4, and R5 are the normal TCP processing steps specified by [RFC0793].

It is important to note that the timestamp **MUST** be checked only when a segment first arrives at the receiver, regardless of whether it is in-sequence or it must be queued for later delivery.

Consider the following example.

Suppose the segment sequence: A.1, B.1, C.1, ..., Z.1 has been sent, where the letter indicates the sequence number and the digit represents the timestamp. Suppose also that segment B.1 has been lost. The timestamp in `TS.Recent` is 1 (from A.1), so C.1, ..., Z.1 are considered acceptable and are queued. When B is retransmitted as segment B.2 (using the latest timestamp), it fills the hole and causes all the segments through Z to be acknowledged and passed to the user. The timestamps of the queued segments are *not* inspected again at this time, since they have already been accepted. When B.2 is accepted, `TS.Recent` is set to 2.

This rule allows reasonable performance under loss. A full window of data is in transit at all times, and after a loss a full window less one segment will show up out-of-sequence to be queued at the receiver (e.g., up to  $2^{30}$  bytes of data); the Timestamps option must not result in discarding this data.

In certain unlikely circumstances, the algorithm of rules R1-R5 could lead to discarding some segments unnecessarily, as shown in the following example:

Suppose again that segments: A.1, B.1, C.1, ..., Z.1 have been sent in sequence and that segment B.1 has been lost. Furthermore, suppose delivery of some of C.1, ... Z.1 is delayed until *\*after\** the retransmission B.2 arrives at the receiver. These delayed segments will be discarded unnecessarily when they do arrive, since their timestamps are now out of date.

This case is very unlikely to occur. If the retransmission was triggered by a timeout, some of the segments C.1, ... Z.1 must have been delayed longer than the RTO time. This is presumably an unlikely event, or there would be many spurious timeouts and retransmissions. If B's retransmission was triggered by the "fast retransmit" algorithm, i.e., by duplicate <ACK>s, then the queued segments that caused these <ACK>s must have been received already.

Even if a segment were delayed past the RTO, the Fast Retransmit mechanism [Jacobson90c] will cause the delayed segments to be retransmitted at the same time as B.2, avoiding an extra RTT and therefore causing a very small performance penalty.

We know of no case with a significant probability of occurrence in which timestamps will cause performance degradation by unnecessarily discarding segments.

#### 5.4. Timestamp Clock

It is important to understand that the PAWS algorithm does not require clock synchronization between sender and receiver. The sender's timestamp clock is used as a source of monotonic non-decreasing values to stamp the segments. The receiver treats the timestamp value as simply a monotonically non-decreasing serial number, without any connection to time. From the receiver's viewpoint, the timestamp is acting as a logical extension of the high-order bits of the sequence number.

The receiver algorithm does place some requirements on the frequency of the timestamp clock.

(a) The timestamp clock must not be "too slow".

It MUST tick at least once for each  $2^{31}$  bytes sent. In fact, in order to be useful to the sender for round trip timing, the clock SHOULD tick at least once per window's worth of data, and even with the window extension defined in Section 2.2,  $2^{31}$

bytes must be at least two windows.

To make this more quantitative, any clock faster than 1 tick/sec will reject old duplicate segments for link speeds of ~8 Gbps. A 1 ms timestamp clock will work at link speeds up to 8 Tbps ( $8 \times 10^{12}$ ) bps!

- (b) The timestamp clock must not be "too fast".

The recycling time of the timestamp clock MUST be greater than MSL seconds. Since the clock (timestamp) is 32 bits and the worst-case MSL is 255 seconds, the maximum acceptable clock frequency is one tick every 59 ns.

However, it is desirable to establish a much longer recycle period, in order to handle outdated timestamps on idle connections (see Section 5.5), and to relax the MSL requirement for preventing sequence number wrap-around. With a 1 ms timestamp clock, the 32-bit timestamp will wrap its sign bit in 24.8 days. Thus, it will reject old duplicates on the same connection if MSL is 24.8 days or less. This appears to be a very safe figure; an MSL of 24.8 days or longer can probably be assumed in the Internet without requiring precise MSL enforcement.

Based upon these considerations, we choose a timestamp clock frequency in the range 1 ms to 1 sec per tick. This range also matches the requirements of the RTTM mechanism, which does not need much more resolution than the granularity of the retransmit timer, e.g., tens or hundreds of milliseconds.

The PAWS mechanism also puts a strong monotonicity requirement on the sender's timestamp clock. The method of implementation of the timestamp clock to meet this requirement depends upon the system hardware and software.

- o Some hosts have a hardware clock that is guaranteed to be monotonic between hardware resets.
- o A clock interrupt may be used to simply increment a binary integer by 1 periodically.
- o The timestamp clock may be derived from a system clock that is subject to being abruptly changed, by adding a variable offset value. This offset is initialized to zero. When a new timestamp clock value is needed, the offset can be adjusted as necessary to make the new value equal to or larger than the previous value (which was saved for this purpose).

- o A random offset may be added to the timestamp clock on a per connection basis. See [RFC6528], section 3, on randomizing the initial sequence number (ISN). The same function with a different secret key can be used to generate the per connection timestamp offset.

### 5.5. Outdated Timestamps

If a connection remains idle long enough for the timestamp clock of the other TCP to wrap its sign bit, then the value saved in TS.Recent will become too old; as a result, the PAWS mechanism will cause all subsequent segments to be rejected, freezing the connection (until the timestamp clock wraps its sign bit again).

With the chosen range of timestamp clock frequencies (1 sec to 1 ms), the time to wrap the sign bit will be between 24.8 days and 24800 days. A TCP connection that is idle for more than 24 days and then comes to life is exceedingly unusual. However, it is undesirable in principle to place any limitation on TCP connection lifetimes.

We therefore require that an implementation of PAWS include a mechanism to "invalidate" the TS.Recent value when a connection is idle for more than 24 days. (An alternative solution to the problem of outdated timestamps would be to send keep-alive segments at a very low rate, but still more often than the wrap-around time for timestamps, e.g., once a day. This would impose negligible overhead. However, the TCP specification has never included keep-alives, so the solution based upon invalidation was chosen.)

Note that a TCP does not know the frequency, and therefore, the wraparound time, of the other TCP, so it must assume the worst. The validity of TS.Recent needs to be checked only if the basic PAWS timestamp check fails, i.e., only if  $SEG.TSval < TS.Recent$ . If TS.Recent is found to be invalid, then the segment is accepted, regardless of the failure of the timestamp check, and rule R3 updates TS.Recent with the TSval from the new segment.

To detect how long the connection has been idle, the TCP MAY update a clock or timestamp value associated with the connection whenever TS.Recent is updated, for example. The details will be implementation-dependent.

### 5.6. Header Prediction

"Header prediction" [Jacobson90a] is a high-performance transport protocol implementation technique that is most important for high-speed links. This technique optimizes the code for the most common case, receiving a segment correctly and in order. Using header

prediction, the receiver asks the question, "Is this segment the next in sequence?" This question can be answered in fewer machine instructions than the question, "Is this segment within the window?"

Adding header prediction to our timestamp procedure leads to the following recommended sequence for processing an arriving TCP segment:

- H1) Check timestamp (same as step R1 above)
- H2) Do header prediction: if segment is next in sequence and if there are no special conditions requiring additional processing, accept the segment, record its timestamp, and skip H3.
- H3) Process the segment normally, as specified in RFC 793. This includes dropping segments that are outside the window and possibly sending acknowledgments, and queuing in-window, out-of-sequence segments.

Another possibility would be to interchange steps H1 and H2, i.e., to perform the header prediction step H2 \*first\*, and perform H1 and H3 only when header prediction fails. This could be a performance improvement, since the timestamp check in step H1 is very unlikely to fail, and it requires unsigned modulo arithmetic. To perform this check on every single segment is contrary to the philosophy of header prediction. We believe that this change might produce a measurable reduction in CPU time for TCP protocol processing on high-speed networks.

However, putting H2 first would create a hazard: a segment from  $2^{32}$  bytes in the past might arrive at exactly the wrong time and be accepted mistakenly by the header-prediction step. The following reasoning has been introduced in [RFC1185] to show that the probability of this failure is negligible.

If all segments are equally likely to show up as old duplicates, then the probability of an old duplicate exactly matching the left window edge is the maximum segment size (MSS) divided by the size of the sequence space. This ratio must be less than  $2^{-16}$ , since MSS must be  $< 2^{16}$ ; for example, it will be  $(2^{12})/(2^{32}) = 2^{-20}$  for a 100 Mbit/s link. However, the older a segment is, the less likely it is to be retained in the Internet, and under any reasonable model of segment lifetime the probability of an old duplicate exactly at the left window edge must be much smaller than  $2^{-16}$ .

The 16 bit TCP checksum also allows a basic unreliability of one part in  $2^{16}$ . A protocol mechanism whose reliability exceeds the

reliability of the TCP checksum should be considered "good enough", i.e., it won't contribute significantly to the overall error rate. We therefore believe we can ignore the problem of an old duplicate being accepted by doing header prediction before checking the timestamp.

However, this probabilistic argument is not universally accepted, and the consensus at present is that the performance gain does not justify the hazard in the general case. It is therefore recommended that H2 follow H1.

#### 5.7. IP Fragmentation

At high data rates, the protection against old segments provided by PAWS can be circumvented by errors in IP fragment reassembly (see [RFC4963]). The only way to protect against incorrect IP fragment reassembly is to not allow the segments to be fragmented. This is done by setting the Don't Fragment (DF) bit in the IP header. Setting the DF bit implies the use of Path MTU Discovery as described in [RFC1191], [RFC1981], and [RFC4821], thus any TCP implementation that implements PAWS MUST also implement Path MTU Discovery.

#### 5.8. Duplicates from Earlier Incarnations of Connection

The PAWS mechanism protects against errors due to sequence number wrap-around on high-speed connections. Segments from an earlier incarnation of the same connection are also a potential cause of old duplicate errors. In both cases, the TCP mechanisms to prevent such errors depend upon the enforcement of a maximum segment lifetime (MSL) by the Internet (IP) layer (see Appendix of RFC 1185 for a detailed discussion). Unlike the case of sequence space wrap-around, the MSL required to prevent old duplicate errors from earlier incarnations does not depend upon the transfer rate. If the IP layer enforces the recommended 2 minute MSL of TCP, and if the TCP rules are followed, TCP connections will be safe from earlier incarnations, no matter how high the network speed. Thus, the PAWS mechanism is not required for this case.

We may still ask whether the PAWS mechanism can provide additional security against old duplicates from earlier connections, allowing us to relax the enforcement of MSL by the IP layer. Appendix B explores this question, showing that further assumptions and/or mechanisms are required, beyond those of PAWS. This is not part of the current extension.

## 6. Conclusions and Acknowledgments

This memo presented a set of extensions to TCP to provide efficient operation over large bandwidth \* delay product paths and reliable operation over very high-speed paths. These extensions are designed to provide compatible interworking with TCP stacks that do not implement the extensions.

These mechanisms are implemented using TCP options for scaled windows and timestamps. The timestamps are used for two distinct mechanisms: RTTM (Round Trip Time Measurement) and PAWS (Protection Against Wrapped Sequences).

The Window Scale option was originally suggested by Mike St. Johns of USAF/DCA. The present form of the option was suggested by Mike Karels of UC Berkeley in response to a more cumbersome scheme defined by Van Jacobson. Lixia Zhang helped formulate the PAWS mechanism description in [RFC1185].

Finally, much of this work originated as the result of discussions within the End-to-End Task Force on the theoretical limitations of transport protocols in general and TCP in particular. Task force members and other on the end2end-interest list have made valuable contributions by pointing out flaws in the algorithms and the documentation. Continued discussion and development since the publication of [RFC1323] originally occurred in the IETF TCP Large Windows Working Group, later on in the End-to-End Task Force, and most recently in the IETF TCP Maintenance Working Group. The authors are grateful for all these contributions.

## 7. Security Considerations

The TCP sequence space is a fixed size, and as the window becomes larger it becomes easier for an attacker to generate forged packets that can fall within the TCP window, and be accepted as valid segments. While use of timestamps and PAWS can help to mitigate this, when using PAWS, if an attacker is able to forge a packet that is acceptable to the TCP connection, a timestamp that is in the future would cause valid segments to be dropped due to PAWS checks. Hence, implementers should take care to not open the TCP window drastically beyond the requirements of the connection.

See [RFC5961] for mitigation strategies to blind in-window attacks.

A naive implementation that derives the timestamp clock value directly from a system uptime clock may unintentionally leak this information to an attacker. This does not directly compromise any of

the mechanisms described in this document. However, this may be valuable information to a potential attacker. It is therefore RECOMMENDED to generate a random, per-connection offset to be used with the clock source when generating the Timestamps option value (see Section 5.4). By carefully choosing this random offset, further improvements as described in [RFC6191] are possible.

Expanding the TCP window beyond 64 KiB for IPv6 allows Jumbograms [RFC2675] to be used when the local network supports packets larger than 64 KiB. When larger TCP segments are used, the TCP checksum becomes weaker.

Mechanisms to protect the TCP header from modification should also protect the TCP options.

Middleboxes and TCP options:

Some middleboxes have been known to remove the TCP options described in this document from TCP segments [Hondall]. Middleboxes that remove TCP options described in this document from the <SYN> segment interfere with the selection of parameters appropriate for the session. Removing any of these options in a <SYN,ACK> segment will leave the end hosts in a state that destroys the proper operation of the protocol.

- \* If a Window Scale option is removed from a <SYN,ACK> segment, the end hosts will not negotiate the window scaling factor correctly. Middleboxes must not remove or modify the Window Scale option from <SYN,ACK> segments.
- \* If a stateful firewall uses the window field to detect whether a received segment is inside the current window, and does not support the Window Scale option, it will not be able to correctly determine whether or not a packet is in the window. These middle boxes must also support the Window Scale option and apply the scale factor when processing segments. If the window scale factor cannot be determined, it must not do window based processing.
- \* If the Timestamps option is removed from the <SYN> or <SYN,ACK> segment, high speed connections that need PAWS would not have that protection. Successful negotiation of Timestamps option enforces a stricter verification of incoming segments at the receiver. If the Timestamps option was removed from a subsequent data segment after a successful negotiation (e.g. as part of re-segmentation), the segment is discarded by the receiver without further processing. Middleboxes should not remove the Timestamps option.

- \* It must be noted that [RFC1323] doesn't address the case of the Timestamps option being dropped or selectively omitted after being negotiated, and that the update in this document may cause some broken middlebox behavior to be detected (potentially unresponsive TCP sessions).

Implementations that depend on PAWS could provide a mechanism for the application to determine whether or not PAWS is in use on the connection, and chose to terminate the connection if that protection doesn't exist. This is not just to protect the connection against middleboxes that might remove the Timestamps option, but also against remote hosts that do not have Timestamp support.

#### 7.1. Privacy Considerations

The TCP options described in this document do not expose individual users data. However, a naive implementation simply using the system clock as source for the Timestamps option will reveal characteristics of the TCP potentially allowing more targeted attacks. It is therefore RECOMMENDED to generate a random, per-connection offset to be used with the clock source when generating the Timestamps option value (see Section 5.4).

Furthermore, the combination, relative ordering and padding of the TCP options described in Section 2.2 and Section 3.2 will reveal additional clues to allow the fingerprinting of the system.

#### 8. IANA Considerations

The described TCP options are well known from the superceded [RFC1323]. IANA is requested to update the "TCP Option Kind Numbers" table under "TCP parameters" to list <this-RFC-to-be> as the reference for the options "WSopt - Window Scale Option" and "TSopt - Timestamps Option".

#### 9. References

##### 9.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

- [RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

## 9.2. Informative References

- [Allman99] Allman, M. and V. Paxson, "On Estimating End-to-End Network Path Properties", Proc. ACM SIGCOMM Technical Symposium, Cambridge, MA, September 1999, <<http://aciri.org/mallman/papers/estimation-la.pdf>>.
- [Floyd05] Floyd, S., "[tcpm] How the RTO should be estimated with timestamps", Message from 26.Jan.2007 to the tcpm mailing list, August 2005, <<http://www.ietf.org/mail-archive/web/tcpm/current/msg02508.html>>.
- [Garlick77] Garlick, L., Rom, R., and J. Postel, "Issues in Reliable Host-to-Host Protocols", Proc. Second Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977, <<http://www.rfc-editor.org/ien/ien12.txt>>.
- [Hondall1] Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., and H. Tokuda, "Is it still possible to extend TCP?", Proc. of ACM Internet Measurement Conference (IMC) '11, November 2011.
- [Jacobson88a] Jacobson, V., "Congestion Avoidance and Control", SIGCOMM '88, Stanford, CA., August 1988, <<http://ee.lbl.gov/papers/congavoid.pdf>>.
- [Jacobson90a] Jacobson, V., "4BSD Header Prediction", ACM Computer Communication Review, April 1990.
- [Jacobson90c] Jacobson, V., "Modified TCP congestion avoidance algorithm", Message to the end2end-interest mailing list, April 1990, <<ftp://ftp.isi.edu/end2end/end2end-interest-1990.mail>>.
- [Karn87] Karn, P. and C. Partridge, "Estimating Round-Trip Times in Reliable Transport Protocols", Proc. SIGCOMM '87, August 1987.

[Kuehlewind10]

Kuehlewind, M. and B. Briscoe, "Chirping for Congestion Control - Implementation Feasibility", November 2010, <[bobbriscoe.net/projects/netsvc\\_i-f/chirp\\_pfldnet10.pdf](http://bobbriscoe.net/projects/netsvc_i-f/chirp_pfldnet10.pdf)>.

[Kuzmanovic03]

Kuzmanovic, A. and E. Knightly, "TCP-LP: Low-Priority Service via End-Point Congestion Control", 2003, <[www.cs.northwestern.edu/~akuzma/doc/TCP-LP-ToN.pdf](http://www.cs.northwestern.edu/~akuzma/doc/TCP-LP-ToN.pdf)>.

[Ludwig00]

Ludwig, R. and K. Sklower, "The Eifel Retransmission Timer", ACM SIGCOMM Computer Communication Review Volume 30 Issue 3, July 2000, <<http://ccr.sigcomm.org/archive/2000/july00/LudwigFinal.pdf>>.

[Martin03]

Martin, D., "[Tsvwg] RFC 1323.bis", Message to the tsvwg mailing list, September 2003, <<http://www.ietf.org/mail-archive/web/tsvwg/current/msg04435.html>>.

[Medina04]

Medina, A., Allman, M., and S. Floyd, "Measuring Interactions Between Transport Protocols and Middleboxes", Proc. ACM SIGCOMM/USENIX Internet Measurement Conference, October 2004, August 2004, <<http://www.icir.net/tbit/tbit-Aug2004.pdf>>.

[Medina05]

Medina, A., Allman, M., and S. Floyd, "Measuring the Evolution of Transport Protocols in the Internet", ACM Computer Communication Review 35(2), April 2005, <<http://icir.net/floyd/papers/TCPEvolution-Mar2005.pdf>>.

[Oppermann13]

Oppermann, A., "[tcpm] Explanation to the relaxation of TSopt acceptance rules", Message to the tcpm mailing list, Jun 2013, <<http://www.ietf.org/mail-archive/web/tcpm/current/msg08001.html>>.

[RFC1072]

Jacobson, V. and R. Braden, "TCP extensions for long-delay paths", RFC 1072, October 1988.

[RFC1122]

Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC1185]

Jacobson, V., Braden, B., and L. Zhang, "TCP Extension for High-Speed Paths", RFC 1185, October 1990.

- [RFC1323] Jacobson, V., Braden, B., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.
- [RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2675] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms", RFC 2675, August 1999.
- [RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.
- [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.
- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.
- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.
- [RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, July 2007.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, August 2010.
- [RFC6191] Gont, F., "Reducing the TIME-WAIT State Using TCP Timestamps", BCP 159, RFC 6191, April 2011.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.
- [RFC6528] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, February 2012.
- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP",

RFC 6675, August 2012.

[RFC6691] Borman, D., "TCP Options and Maximum Segment Size (MSS)", RFC 6691, July 2012.

[RFC6817] Shalunov, S., Hazel, G., Iyengar, J., and M. Kuehlewind, "Low Extra Delay Background Transport (LEDBAT)", RFC 6817, December 2012.

## Appendix A. Implementation Suggestions

### TCP Option Layout

The following layout is recommended for sending options on non-<SYN> segments, to achieve maximum feasible alignment of 32-bit and 64-bit machines.

```

+-----+-----+-----+-----+
|  NOP  |  NOP  | TSopt |  10  |
+-----+-----+-----+-----+
|                TSval timestamp                |
+-----+-----+-----+-----+
|                TSecr timestamp                |
+-----+-----+-----+-----+

```

### Interaction with the TCP Urgent Pointer

The TCP Urgent pointer, like the TCP window, is a 16 bit value. Some of the original discussion for the TCP Window Scale option included proposals to increase the Urgent pointer to 32 bits. As it turns out, this is unnecessary. There are two observations that should be made:

- (1) With IP Version 4, the largest amount of TCP data that can be sent in a single packet is 65495 bytes (64 KiB - 1 -- size of fixed IP and TCP headers).
- (2) Updates to the urgent pointer while the user is in "urgent mode" are invisible to the user.

This means that if the Urgent Pointer points beyond the end of the TCP data in the current segment, then the user will remain in urgent mode until the next TCP segment arrives. That segment will update the urgent pointer to a new offset, and the user will never have left urgent mode.

Thus, to properly implement the Urgent Pointer, the sending TCP only has to check for overflow of the 16 bit Urgent Pointer field before filling it in. If it does overflow, then a value of 65535 should be inserted into the Urgent Pointer.

The same technique applies to IP Version 6, except in the case of IPv6 Jumbograms. When IPv6 Jumbograms are supported, [RFC2675] requires additional steps for dealing with the Urgent Pointer, these are described in section 5.2 of [RFC2675].

## Appendix B. Duplicates from Earlier Connection Incarnations

There are two cases to be considered: (1) a system crashing (and losing connection state) and restarting, and (2) the same connection being closed and reopened without a loss of host state. These will be described in the following two sections.

### B.1. System Crash with Loss of State

TCP's quiet time of one MSL upon system startup handles the loss of connection state in a system crash/restart. For an explanation, see for example "When to Keep Quiet" in the TCP protocol specification [RFC0793]. The MSL that is required here does not depend upon the transfer speed. The current TCP MSL of 2 minutes seemed acceptable as an operational compromise, when many host systems used to take this long to boot after a crash. Current host systems can boot considerably faster.

The Timestamps option may be used to ease the MSL requirements (or to provide additional security against data corruption). If timestamps are being used and if the timestamp clock can be guaranteed to be monotonic over a system crash/restart, i.e., if the first value of the sender's timestamp clock after a crash/restart can be guaranteed to be greater than the last value before the restart, then a quiet time is unnecessary.

To dispense totally with the quiet time would require that the host clock be synchronized to a time source that is stable over the crash/restart period, with an accuracy of one timestamp clock tick or better. We can back off from this strict requirement to take advantage of approximate clock synchronization. Suppose that the clock is always re-synchronized to within N timestamp clock ticks and that booting (extended with a quiet time, if necessary) takes more than N ticks. This will guarantee monotonicity of the timestamps, which can then be used to reject old duplicates even without an enforced MSL.

## B.2. Closing and Reopening a Connection

When a TCP connection is closed, a delay of  $2 \times \text{MSL}$  in TIME-WAIT state ties up the socket pair for 4 minutes (see Section 3.5 of [RFC0793]). Applications built upon TCP that close one connection and open a new one (e.g., an FTP data transfer connection using Stream mode) must choose a new socket pair each time. The TIME-WAIT delay serves two different purposes:

- (a) Implement the full-duplex reliable close handshake of TCP.

The proper time to delay the final close step is not really related to the MSL; it depends instead upon the RTT for the FIN segments and therefore upon the RTT of the path. (It could be argued that the side that is sending a FIN knows what degree of reliability it needs, and therefore it should be able to determine the length of the TIME-WAIT delay for the FIN's recipient. This could be accomplished with an appropriate TCP option in FIN segments.)

Although there is no formal upper-bound on RTT, common network engineering practice makes an RTT greater than 1 minute very unlikely. Thus, the 4 minute delay in TIME-WAIT state works satisfactorily to provide a reliable full-duplex TCP close. Note again that this is independent of MSL enforcement and network speed.

The TIME-WAIT state could cause an indirect performance problem if an application needed to repeatedly close one connection and open another at a very high frequency, since the number of available TCP ports on a host is less than  $2^{16}$ . However, high network speeds are not the major contributor to this problem; the RTT is the limiting factor in how quickly connections can be opened and closed. Therefore, this problem will be no worse at high transfer speeds.

- (b) Allow old duplicate segments to expire.

To replace this function of TIME-WAIT state, a mechanism would have to operate across connections. PAWS is defined strictly within a single connection; the last timestamp (TS.Recent) is kept in the connection control block, and discarded when a connection is closed.

An additional mechanism could be added to the TCP, a per-host cache of the last timestamp received from any connection. This value could then be used in the PAWS mechanism to reject old duplicate segments from earlier incarnations of the connection,

if the timestamp clock can be guaranteed to have ticked at least once since the old connection was open. This would require that the TIME-WAIT delay plus the RTT together must be at least one tick of the sender's timestamp clock. Such an extension is not part of the proposal of this RFC.

Note that this is a variant on the mechanism proposed by Garlick, Rom, and Postel [Garlick77], which required each host to maintain connection records containing the highest sequence numbers on every connection. Using timestamps instead, it is only necessary to keep one quantity per remote host, regardless of the number of simultaneous connections to that host.

## Appendix C. Summary of Notation

The following notation has been used in this document.

### Options

WSopt:	TCP Window Scale option
TSopt:	TCP Timestamps option

### Option Fields

shift.cnt:	Window scale byte in WSopt
TSval:	32-bit Timestamp Value field in TSopt
TSecr:	32-bit Timestamp Reply field in TSopt

### Option Fields in Current Segment

SEG.TSval:	TSval field from TSopt in current segment
SEG.TSecr:	TSecr field from TSopt in current segment
SEG.WSopt:	8-bit value in WSopt

### Clock Values

my.TSclock:	System wide source of 32-bit timestamp values
my.TSclock.rate:	Period of my.TSclock (1 ms to 1 sec)
Snd.TSoffset:	A offset for randomizing Snd.TSclock
Snd.TSclock:	my.TSclock + Snd.TSoffset

### Per-Connection State Variables

TS.Recent:            Latest received Timestamp  
Last.ACK.sent:        Last ACK field sent  
Snd.TS.OK:            1-bit flag  
Snd.WS.OK:            1-bit flag  
Rcv.Wind.Shift:       Receive window scale exponent  
Snd.Wind.Shift:       Send window scale exponent  
Start.Time:           Snd.TSclock value when segment being timed was  
                      sent (used by pre-1323 code).

#### Procedure

Update\_SRTT(m)       Procedure to update the smoothed RTT and RTT  
                      variance estimates, using the rules of  
                      [Jacobson88a], given m, a new RTT measurement

#### Appendix D. Event Processing Summary

##### OPEN Call

...

An initial send sequence number (ISS) is selected. Send a <SYN> segment of the form:

<SEQ=ISS><CTL=SYN><TSval=Snd.TSclock><WSopt=Rcv.Wind.Shift>

...

##### SEND Call

CLOSED STATE (i.e., TCB does not exist)

...

##### LISTEN STATE

If the foreign socket is specified, then change the connection from passive to active, select an ISS. Send a <SYN> segment containing the options: <TSval=Snd.TSclock> and <WSopt=Rcv.Wind.Shift>. Set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. ...

##### SYN-SENT STATE

##### SYN-RECEIVED STATE

...

ESTABLISHED STATE  
CLOSE-WAIT STATE

Segmentize the buffer and send it with a piggybacked acknowledgment (acknowledgment value = RCV.NXT). ...

If the urgent flag is set ...

If the Snd.TS.OK flag is set, then include the TCP Timestamps option <TSval=Snd.TSclock,TSecr=TS.Recent> in each data segment.

Scale the receive window for transmission in the segment header:

SEG.WND = (RCV.WND >> Rcv.Wind.Shift).

SEGMENT ARRIVES

...

If the state is LISTEN then

first check for an RST

...

second check for an ACK

...

third check for a SYN

if the SYN bit is set, check the security. If the ...

...

if the SEG.PRC is less than the TCB.PRC then continue.

Check for a Window Scale option (WSopt); if one is found, save SEG.WSopt in Snd.Wind.Shift and set Snd.WS.OK flag on. Otherwise, set both Snd.Wind.Shift and Rcv.Wind.Shift to zero and clear Snd.WS.OK flag.

Check for a TSopt option; if one is found, save SEG.TSval in the variable TS.Recent and turn on the Snd.TS.OK bit.

Set RCV.NXT to SEG.SEQ+1, IRS is set to SEG.SEQ and any other control or text should be queued for processing later. ISS should be selected and a <SYN> segment sent of the form:

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

If the Snd.WS.OK bit is on, include a WSopt option <WSopt=Rcv.Wind.Shift> in this segment. If the Snd.TS.OK bit is on, include a TSopt <TSval=Snd.TSclock, TSecr=TS.Recent> in this segment. Last.ACK.sent is set to RCV.NXT.

SND.NXT is set to ISS+1 and SND.UNA to ISS. The connection state should be changed to SYN-RECEIVED. Note that any other incoming control or data (combined with SYN) will be processed in the SYN-RECEIVED state, but processing of SYN and ACK should not be repeated. If the listen was not fully specified (i.e., the foreign socket was not fully specified), then the unspecified fields should be filled in now.

fourth other text or control

...

If the state is SYN-SENT then

first check the ACK bit

...

...

fourth check the SYN bit

...

If the SYN bit is on and the security/compartments and precedence are acceptable then, RCV.NXT is set to SEG.SEQ+1, IRS is set to SEG.SEQ, and any acknowledgments on the retransmission queue which are thereby acknowledged should be removed.

Check for a Window Scale option (WSopt); if it is found, save SEG.WSopt in Snd.Wind.Shift; otherwise, set both Snd.Wind.Shift and Rcv.Wind.Shift to zero.

Check for a TSopt option; if one is found, save SEG.TSval in variable TS.Recent and turn on the Snd.TS.OK bit in the connection control block. If the ACK bit is set, use Snd.TSclock - SEG.TSecr as the initial RTT estimate.

If SND.UNA > ISS (our <SYN> has been ACKed), change the connection state to ESTABLISHED, form an <ACK> segment:

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

and send it. If the Snd.Echo.OK bit is on, include a TSopt option <TSval=Snd.TSclock,TSecr=TS.Recent> in this <ACK> segment. Last.ACK.sent is set to RCV.NXT.

Data or controls which were queued for transmission may be included. If there are other controls or text in the segment then continue processing at the sixth step below where the URG bit is checked, otherwise return.

Otherwise enter SYN-RECEIVED, form a <SYN,ACK> segment:

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

and send it. If the Snd.Echo.OK bit is on, include a TSopt option <TSval=Snd.TSclock,TSecr=TS.Recent> in this segment. If the Snd.WS.OK bit is on, include a WSopt option <WSopt=Rcv.Wind.Shift> in this segment. Last.ACK.sent is set to RCV.NXT.

If there are other controls or text in the segment, queue them for processing after the ESTABLISHED state has been reached, return.

fifth, if neither of the SYN or RST bits is set then drop the segment and return.

Otherwise,

First, check sequence number

SYN-RECEIVED STATE  
ESTABLISHED STATE  
FIN-WAIT-1 STATE  
FIN-WAIT-2 STATE  
CLOSE-WAIT STATE  
CLOSING STATE  
LAST-ACK STATE  
TIME-WAIT STATE

Segments are processed in sequence. Initial tests on arrival are used to discard old duplicates, but further processing is done in SEG.SEQ order. If a segment's contents straddle the boundary between old and new, only the new parts should be processed.

Rescale the received window field:

TrueWindow = SEG.WND << Snd.Wind.Shift,

and use "TrueWindow" in place of SEG.WND in the following steps.

Check whether the segment contains a Timestamps option and bit Snd.TS.OK is on. If so:

If SEG.TSval < TS.Recent and the RST bit is off, then test whether connection has been idle less than 24 days; if all are true, then the segment is not acceptable; follow steps below for an unacceptable segment.

If SEG.SEQ is less than or equal to Last.ACK.sent, then save SEG.TSval in variable TS.Recent.

There are four cases for the acceptability test for an incoming segment:

...

If an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set, if so drop the segment and return):

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

Last.ACK.sent is set to SEG.ACK of the acknowledgment. If the Snd.Echo.OK bit is on, include the Timestamps option <TSval=Snd.TSclock,TSecr=TS.Recent> in this <ACK> segment. Set Last.ACK.sent to SEG.ACK and send the <ACK> segment. After sending the acknowledgment, drop the unacceptable segment and return.

...

fifth check the ACK field.

if the ACK bit is off drop the segment and return.

if the ACK bit is on

...

ESTABLISHED STATE

If SND.UNA < SEG.ACK <= SND.NXT then, set SND.UNA <- SEG.ACK. Also compute a new estimate of round-trip time. If Snd.TS.OK bit is on, use Snd.TSclock - SEG.TSecr; otherwise use the elapsed time since the first segment in the retransmission queue was sent. Any segments on the retransmission queue which are thereby entirely acknowledged...

...

Seventh, process the segment text.

ESTABLISHED STATE

FIN-WAIT-1 STATE

FIN-WAIT-2 STATE

...

Send an acknowledgment of the form:

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

If the Snd.TS.OK bit is on, include Timestamps option <TSval=Snd.TSclock,TSecr=TS.Recent> in this <ACK> segment. Set Last.ACK.sent to SEG.ACK of the acknowledgment, and send it. This acknowledgment should be piggy-backed on a segment being transmitted if possible without incurring undue delay.

...

#### Appendix E. Timestamps Edge Cases

While the rules laid out for when to calculate RTTM produce the correct results most of the time, there are some edge cases where an incorrect RTTM can be calculated. All of these situations involve the loss of segments. It is felt that these scenarios are rare, and that if they should happen, they will cause a single RTTM measurement to be inflated, which mitigates its effects on RTO calculations.

[Martin03] cites two similar cases when the returning <ACK> is lost, and before the retransmission timer fires, another returning <ACK>

segment arrives, which acknowledges the data. In this case, the RTTM calculated will be inflated:

```

clock
tc=1    <A, TSval=1> ----->

tc=2    (lost) <---- <ACK(A), TSecr=1, win=n>
        (RTTM would have been 1)

        (receive window opens, window update is sent)
tc=5    <---- <ACK(A), TSecr=1, win=m>
        (RTTM is calculated at 4)

```

One thing to note about this situation is that it is somewhat bounded by RTO + RTT, limiting how far off the RTTM calculation will be. While more complex scenarios can be constructed that produce larger inflations (e.g., retransmissions are lost), those scenarios involve multiple segment losses, and the connection will have other more serious operational problems than using an inflated RTTM in the RTO calculation.

#### Appendix F. Window Retraction Example

Consider an established TCP connection using a scale factor of 128, Snd.Wind.Shift=7 and Rcv.Wind.Shift=7, that is running with a very small window because the receiver is bottlenecked and both ends are doing small reads and writes.

Consider the ACKs coming back:

SEG.ACK	SEG.WIN	computed	SND.WIN	receiver's actual window
1000	2	1256		1300

The sender writes 40 bytes and receiver ACKs:

1040	2	1296		1300
------	---	------	--	------

The sender writes 5 additional bytes and the receiver has a problem. Two choices:

1045	2	1301		1300	- BEYOND BUFFER
1045	1	1173		1300	- RETRACTED WINDOW

This is a general problem and can happen any time the sender does a write which is smaller than the window scale factor.

In most stacks it is at least partially obscured when the window size is larger than some small number of segments because the stacks prefer to announce windows that are an integral number of segments, rounded up to the next scale factor. This plus silly window suppression tends to cause less frequent, larger window updates. If the window was rounded down to a segment size there is more opportunity to advance the window, the BEYOND BUFFER case above, rather than retracting it.

#### Appendix G. RTO calculation modification

Taking multiple RTT samples per window would shorten the history calculated by the RTO mechanism in [RFC6298], and the below algorithm aims to maintain a similar history as originally intended by [RFC6298].

It is roughly known how many samples a congestion window worth of data will yield, not accounting for ACK compression, and ACK losses. Such events will result in more history of the path being reflected in the final value for RTO, and are uncritical. This modification will ensure that a similar amount of time is taken into account for the RTO estimation, regardless of how many samples are taken per window:

```
ExpectedSamples = ceiling(FlightSize / (SMSS * 2))
```

```
alpha' = alpha / ExpectedSamples
```

```
beta' = beta / ExpectedSamples
```

Note that the factor 2 in ExpectedSamples is due to "Delayed ACKs".

Instead of using alpha and beta in the algorithm of [RFC6298], use alpha' and beta' instead:

```
RTTVAR <- (1 - beta') * RTTVAR + beta' * |SRTT - R'|
```

```
SRTT <- (1 - alpha') * SRTT + alpha' * R'
```

```
(for each sample R')
```

#### Appendix H. Changes from RFC 1323

Several important updates and clarifications to the specification in RFC 1323 are made in these document. The technical changes are summarized below:

- (a) A wrong reference to SND.WND was corrected to SEG.WND in Section 2.3
- (b) Section 2.4 was added describing the unavoidable window retraction issue, and explicitly describing the mitigation steps necessary.
- (c) In Section 3.2 the wording how the Timestamps option negotiation is to be performed was updated with RFC2119 wording. Further, a number of paragraphs were added to clarify the expected behavior with a compliant implementation using TSopt, as RFC1323 left room for interpretation - e.g. potential late enablement of TSopt.
- (d) The description of which TSecr values can be used to update the measured RTT has been clarified. Specifically, with timestamps, the Karn algorithm [Karn87] is disabled. The Karn algorithm disables all RTT measurements during retransmission, since it is ambiguous whether the <ACK> is for the original segment, or the retransmitted segment. With timestamps, that ambiguity is removed since the TSecr in the <ACK> will contain the TSval from whichever data segment made it to the destination.
- (e) RTTM update processing explicitly excludes segments not updating SND.UNA. The original text could be interpreted to allow taking RTT samples when SACK acknowledges some new, non-continuous data.
- (f) In RFC1323, section 3.4, step (2) of the algorithm to control which timestamp is echoed was incorrect in two regards:
  - (1) It failed to update TS.recent for a retransmitted segment that resulted from a lost <ACK>.
  - (2) It failed if SEG.LEN = 0.In the new algorithm, the case of SEG.TSval >= TS.recent is included for consistency with the PAWS test.
- (g) It is now recommended that the Timestamps option is included in <RST> segments if the incoming segment contained a Timestamps option.
- (h) <RST> segments are explicitly excluded from PAWS processing.

- (i) Added text to clarify the precedence between regular TCP [RFC0793] and this document Timestamps option / PAWS processing. Discussion about combined acceptability checks are ongoing.
- (j) Snd.TSoffset and Snd.TSclock variables have been added. Snd.TSclock is the sum of my.TSclock and Snd.TSoffset. This allows the starting points for timestamp values to be randomized on a per-connection basis. Setting Snd.TSoffset to zero yields the same results as [RFC1323]. Text was added to guide implementers to the proper selection of these offsets, as entirely random offsets for each new connection will conflict with PAWS.
- (k) Appendix A has been expanded with information about the TCP Urgent Pointer. An earlier revision contained text around the TCP MSS option, which was split off into [RFC6691].
- (l) One correction was made to the Event Processing Summary in Appendix D. In SEND CALL/ESTABLISHED STATE, RCV.WND is used to fill in the SEG.WND value, not SND.WND.
- (m) Appendix G was added to exemplify how an RTO calculation might be updated to properly take the much higher RTT sampling frequency enabled by the Timestamps option into account.

Editorial changes of the document, that don't impact the implementation or function of the mechanisms described in this document include:

- (a) Removed much of the discussion in Section 1 to streamline the document. However, detailed examples and discussions in Section 2, Section 3 and Section 5 are kept as guideline for implementers.
- (b) Added short text that the use of WS increases the chances of sequence number wrap, thus the PAWS mechanism is required in certain environments.
- (c) Removed references to "new" options, as the options were introduced in [RFC1323] already. Changed the text in Section 1.3 to specifically address TS and WS options.
- (d) Section 1.4 was added for [RFC2119] wording. Normative text was updated with the appropriate phrases.

- (e) Added < > brackets to mark specific types of segments, and replaced most occurrences of "packet" with "segment", where TCP segments are referred to.
- (f) Updated the text in Section 3 to take into account what has been learned since [RFC1323].
- (g) Removed some unused references.
- (h) Removed the list of changes between [RFC1323] and prior versions. These changes are mentioned in Appendix C of [RFC1323].
- (i) Moved Appendix Changes from RFC 1323 to the end of the appendices for easier lookup. In addition, the entries were split into a technical and an editorial part, and sorted to roughly correspond with the sections in the text where they apply.

#### Authors' Addresses

David Borman  
Quantum Corporation  
Mendota Heights MN 55120  
USA

Email: david.borman@quantum.com

Bob Braden  
University of Southern California  
4676 Admiralty Way  
Marina del Rey CA 90292  
USA

Email: braden@isi.edu

Van Jacobson  
Google, Inc.  
1600 Amphitheatre Parkway  
Mountain View CA 94043  
USA

Email: vanj@google.com

Richard Scheffenegger (editor)  
NetApp, Inc.  
Am Euro Platz 2  
Vienna, 1120  
Austria

Email: [rs@netapp.com](mailto:rs@netapp.com)



TCPM Working Group  
Internet Draft  
Intended status: Proposed Standard  
Expires: December 2013

J. Touch  
USC/ISI  
June 4, 2013

Shared Use of Experimental TCP Options  
draft-ietf-tcpm-experimental-options-06.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on December 4, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in

Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Abstract

This document describes how the experimental TCP option codepoints can concurrently support multiple TCP extensions, even within the same connection. It uses a new IANA TCP experiment identifier, and is also robust to experiments that are not registered and those that do not use this sharing mechanism. It is recommended for all new TCP options that use these codepoints.

## Table of Contents

1. Introduction.....	2
2. Conventions used in this document.....	4
3. TCP Experimental Option Structure.....	4
3.1. Selecting an ExID.....	6
3.2. Impact on TCP Option Processing.....	7
4. Reducing the Impact of False Positives.....	7
5. Migration to Assigned Options.....	8
6. Rationale.....	8
7. Security Considerations.....	9
8. IANA Considerations.....	9
9. References.....	10
9.1. Normative References.....	10
9.2. Informative References.....	10
10. Acknowledgments.....	12

## 1. Introduction

TCP includes options to enable new protocol capabilities that can be activated only where needed and supported [RFC793]. The space for identifying such options is small - 256 values, of which 30 are assigned at the time this document was published [IANA]. Two of these codepoints are allocated to support experiments (253, 254) [RFC4727]. These values are intended for testing purposes or anytime an assigned codepoint is either not warranted or available, e.g., based on the maturity status of the defined capability (i.e., Experimental or Informational, rather than Standards Track).

The term "experimental TCP options" refers here to options that use the TCP experimental option codepoints [RFC4727]. Such experiments can be described in any type of RFC - Experimental, Informational, etc., and are intended to be used both in controlled environments

and in are allowed in public deployments (when not enabled as default) [RFC3692]. Nothing prohibits deploying multiple experiments in the same environment - controlled or public. Further, some protocols are specified in Experimental or Informational RFCs, which either include parameters or design choices not yet understood or which might not be widely deployed [RFC2026]. TCP options in such RFCs are typically not eligible for assigned TCP option codepoints [RFC2780], and so there is a need to share use of the experimental option codepoints.

There is currently no mechanism to support shared use of the TCP experimental option codepoints, either by different experiments on different connections, or for more than two experimental options in the same connection. Experimental options 253 and 254 are already deployed in operational code to support an early version of TCP authentication. Option 253 is also documented for the experimental TCP Cookie Transaction option [RFC6013]. This shared use results in collisions in which a single codepoint can appear multiple times in a single TCP segment and for which each use is ambiguous.

Other codepoints have been used without assignment (known as "squatting"), notably 31-32 (TCP cookie transactions, as originally distributed and in its API doc) and 76-78 (tcpcrypt) [Bill][Sill]. Commercial products reportedly also use unassigned options 33, 69-70, and 76-78 as well. Even though these uses are unauthorized, they currently impact legitimate assignees.

Both such misuses (squatting on both experimental and assigned codepoints) are expected to continue, but there are several approaches which can alleviate the impact on cooperating protocol designers. One proposal relaxes the requirements for assignment of TCP options, allowing them to be assigned more readily for protocols that have not been standardized through the IETF process [RFC5226]. Another proposal assigns a larger pool to the TCP experiment option codepoints and manages their sharing through IANA coordination [Ed11].

The approach proposed in this document does not require additional TCP option codepoints, and is robust to those who choose either not to support it or not to register their experiments. The solution adds a field to the structure of the experimental TCP option. This field is populated with an "experiment identifier" (ExID) defined as part of a specific option experiment. The ExID helps reduce the probability of a collision of independent experimental uses of the same option codepoint, both for those who follow this document (using registered ExIDs) and those who do not (squatters who either ignore this extension or do not register their ExIDs).

The solution proposed in this document is recommended for all new protocols that use TCP experimental option codepoints. The techniques used here may also help share other experimental codepoints, but that issue is out of scope for this document.

## 2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

## 3. TCP Experimental Option Structure

TCP options have the current common structure [RFC793], in which the first byte is the codepoint (Kind) and the second byte is the length of the option in bytes (Length):

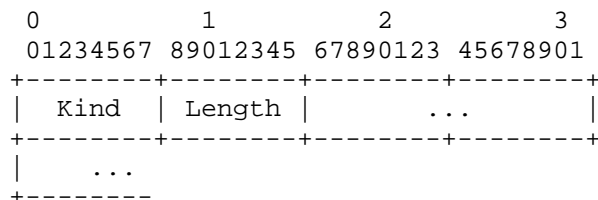


Figure 1 TCP Option Structure [RFC793]

This document extends the option structure for experimental codepoints (253, 254) with an experiment identifier (ExID), which is either 2 or 4 bytes in length. The ExID is used to differentiate different experiments, and is the first field after the Kind and Length, as follows:

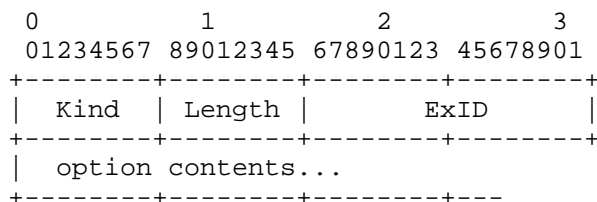


Figure 2 TCP Experimental Option with a 16-bit ExID

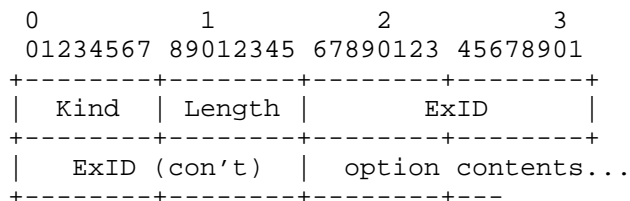


Figure 3 TCP Experimental Option with a 32-bit ExID

This mechanism is encouraged for all TCP options that are not yet eligible for assigned codepoints:

>> Protocols requiring new TCP option codepoints that are not eligible for assigned values SHOULD use the existing TCP experimental option codepoints (253, 254) with ExIDs as described in this document.

This mechanism is encouraged for all TCP options using the current experimental codepoints in controlled environments:

>> All protocols using the TCP experimental option codepoints (253, 254), even those deployed in controlled environments, SHOULD use ExIDs as described in this document.

This mechanism is required for all TCP options using the current experimental codepoints that are publicly deployed, whether enabled by default or not:

>> All protocols using the TCP experimental option codepoints (253, 254) that are deployed outside controlled environments, such as in the public Internet, MUST use ExIDs as described in this document.

Once a TCP option uses the mechanism in this document, registration of the ExID with IANA is required:

>> All protocols using ExIDs as described in this document MUST register those ExIDs with IANA.

Applicants register their desired ExID by contacting IANA [IANA].

### 3.1. Selecting an ExID

ExIDs are selected at design time, when the protocol designer first implements or specifies the experimental option. ExIDs can be either 16-bits or 32-bits. In both cases, the value is stored in the header in network-standard (big-endian) byte order. ExIDs combine properties of IANA registered codepoints with "magic numbers".

>> All ExIDs MUST be either 16-bits or 32-bits long.

Use of the ExID, whether 16-bit or 32-bit, helps reduce the probability of a false positive collision with those who either do not register their experiment or who do not implement this mechanism.

In order to conserve TCP option space, either for use within a specific option or to be available for other options:

>> Options implementing the mechanism of this document SHOULD use 16-bit ExIDs except where explicitly motivating the need for 32-bit ExIDs, e.g., to avoid false positives or maintain alignment with an expected future assigned codepoint.

ExIDs are registered with IANA using "first-come, first-served" priority based on the first two bytes. Those two bytes are thus sufficient to interpret which experimental option is contained in the option field.

>> All ExIDs MUST be unique based on their first 16 bits.

The second two bytes serve as a "magic number". A magic number is a self-selected codepoint whose primary value is its unlikely collision with values selected by others. Magic numbers are used in other protocols, e.g., BOOTP [RFC951] and DHCP [RFC2131].

Using the additional magic number bytes helps the option contents have the same byte alignment in the TCP header as they would have if (or when) a conventional (non-experiment) TCP option codepoint is assigned. Use of the same alignment reduces the potential for implementation errors, especially in using the same word-alignment padding, if the same software is later modified to use a conventional codepoint. Use of the longer, 32-bit ExID further

decreases the probability of such a false positive compared to those using shorter, 16-bit ExIDs.

Use of the ExID does consume TCP option space but enables concurrent use of the experimental codepoints and provides protection against false positives, leaving less space for other options (including other experiments). Use of the longer, 32-bit ExID consumes more space, but provides more protection against false positives.

### 3.2. Impact on TCP Option Processing

The ExID number is considered part of the TCP option, not the TCP option header. The presence of the ExID increases the effective option Length field by the size of the ExID. The presence of this ExID is thus transparent to implementations that do not support TCP options where it is used.

During TCP processing, ExIDs in experimental options are matched against the ExIDs for each implemented protocol. The remainder of the option is specified by the particular experimental protocol.

>> Experimental options that have ExIDs that do not match implemented protocols MUST be ignored.

The ExID mechanism must be coordinated during connection establishment, just as with any TCP option.

>> TCP ExID, if used in any TCP segment of a connection, MUST be present in TCP SYN segments of that connection.

>> TCP experimental option ExIDs, if used in any TCP segment of a connection, SHOULD be used in all TCP segments of that connection in which any experimental option is present.

Use of an ExID uses additional space in the TCP header and requires additional protocol processing by experimental protocols. Because these are experiments, neither consideration is a substantial impediment; a finalized protocol can avoid both issues with the assignment of a dedicated option codepoint later.

### 4. Reducing the Impact of False Positives

False positives occur where the registered ExID of an experiment matches the value of an option that does not use ExIDs. Such collisions can cause an option to be interpreted by the incorrect processing routine. Use of checksums or signatures may help an

experiment use the shorter ExID while reducing the corresponding increased potential for false positives.

>> Experiments that are not robust to ExID false positives SHOULD implement other detection measures, such as checksums or minimal digital signatures over the experimental options they support.

## 5. Migration to Assigned Options

Some experiments may transition from experiment, and become eligible for an assigned TCP option codepoint. This document does not recommend a specific migration plan to transition from use of the experimental TCP options/ExIDs to use of an assigned codepoint.

However, once an assigned codepoint is allocated, use of an ExID represents unnecessary overhead. As a result:

>> Once a TCP option codepoint is assigned to a protocol, that protocol SHOULD NOT continue to use an ExID as part of that assigned codepoint.

This document does not recommend whether or how an implementation of an assigned codepoint can be backward-compatible with use of the experimental codepoint/ExID.

However, some implementers may be tempted to include both the experimental and assigned codepoint in the same segment, e.g., in a SYN to support backward-compatibility during connection establishment. This is a poor use limited resources and so to ensure conservation of the TCP option space:

>> A TCP segment MUST NOT contain both an assigned TCP option codepoint and a TCP experimental option codepoint for the same protocol.

Instead, a TCP that intends backward compatibility might send multiple SYNs with alternates of the same option and discard all but the most desired successful connection. Although this approach may resolve more slowly or require additional effort at the endpoints, it is preferable to excessively consuming TCP option space.

## 6. Rationale

The ExIDs described in this document combine properties of IANA first-come/first-served (FCFS) registered values with magic numbers. Although IANA FCFS registries are common, so too are those who either fail to register or who 'squat' by deliberately using

codepoints that are assigned to others. The approach in this document is intended to recognize this reality and be more robust to its consequences than would be a conventional IANA FCFS registry.

Existing ID spaces were considered as ExIDs in the development of this mechanism, including IEEE Organizationally Unique Identifier (OUI) and IANA Private Enterprise Numbers (PENs) [802] [OUI] [RFC1155].

OUIs are 24-bit identifiers that are combined with 24 to 40-bits of privately-assigned space to create identifiers that commonly assigned to a unique piece of hardware. OUIs are already longer than the smaller ExID value, and obtaining an OUI is costly (currently \$1,885.00 USD). An OUI could be obtained for each experiment, but this could be considered expensive. An OUI already assigned to an organization could be shared if extended (to support multiple experiments within an organization), but this would either require coordination within an organization or an IANA registry; the former is prohibitive, and the latter is more complicated than to have IANA manage the entire space.

PENs were originally used in SNMP [RFC1157]. PENs are identifiers that can be obtained without cost from IANA [PEN]. Despite the current registry, the size of the PEN assignment space is currently undefined, and has only recently been proposed (as 32-bits) [Lil2]. PENs are currently assigned to organizations, and there is no current process for assigning them to individuals. Finally, if 32-bits as expected, they would be larger than needed in many cases.

## 7. Security Considerations

The mechanism described in this document is not intended to provide (nor does it weaken existing) security for TCP option processing.

## 8. IANA Considerations

This document calls for IANA to create a new TCP experimental option Experiment Identifier (ExID) registry. The registry records both 16-bit and 32-bit ExIDs, as well as a name and e-mail contact for each entry. ExIDs are registered for use with both TCP experimental option codepoints, i.e., with TCP options with values of 253 and 254.

Entries are assigned on a First-Come, First-Served (FCFS) basis [RFC5226]. The registry operates FCFS on the first two bytes of the ExID (in network-standard order) but records the entire ExID (in network-standard order). Some examples are:

- o 0x12340000 collides with a previous registration of 0x1234abcd
- o 0x5678 collides with a previous registration of 0x56780123
- o 0xabcd1234 collides it a previous registration of 0xabcd

IANA will advise applicants of duplicate entries to select an alternate value, as per typical FCFS processing.

IANA will record known duplicate uses to assist the community in both debugging assigned uses as well as correcting unauthorized duplicate uses.

IANA should impose no requirements on making a registration other than indicating the desired codepoint and providing a point of contact. A short description or acronym for the use is desired, but should not be required.

## 9. References

### 9.1. Normative References

- [RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, Sep. 1981.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4727] Fenner, B., "Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers", RFC 4727, Nov. 2006.
- [RFC5226] Narten, T., H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

### 9.2. Informative References

- [802] "IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture", IEEE 802-2001, 8 March 2002.
- [Bill] Bittau, A., D. Boneh, M. Hamburg, M. Handley, D. Mazieres, Q. Slack, "Cryptographic protection of TCP Streams (tcpcrypt)", work in progress, draft-bittau-tcp-crypt-03, Sep. 3, 2012.

- [Ed11] Eddy, W., "Additional TCP Experimental-Use Options", work in progress, draft-eddy-tcpm-addl-exp-options-00, Aug. 16, 2011.
- [IANA] IANA web pages, <http://www.iana.org/>
- [Lil12] Liang, P., A. Melnikov, "Private Enterprise Number (PEN) practices and Internet Assigned Numbers: Authority (IANA) considerations for registration procedures", draft-liang-iana-pen-01, (work in progress), June 2012.
- [OUI] IEEE OUI registry, <http://standards.ieee.org/develop/regauth/oui/>
- [PEN] IANA Private Enterprise Numbers, <http://www.iana.org/assignments/enterprise-numbers>
- [RFC951] Croft, B., J. Gilmore, "BOOTSTRAP PROTOCOL (BOOTP)", RFC 951, Sept. 1985.
- [RFC1155] Rose, M., K. McCloghrie, "Structure and Identification of Management Information for TCP/IP-based internets", RFC 1155, May 1990.
- [RFC1157] Case, J., M. Fedor, M. Schoffstall, J. Davin, "A Simple Network Management Protocol (SNMP)", RFC 1157, May 1990.
- [RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, Oct. 1996.
- [RFC2131] Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, Mar. 1997.
- [RFC2780] Bradner, S., V. Paxson, "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers", BCP 37, RFC 2780, Mar. 2000.
- [RFC3692] Narten, T., "Assigning Experimental and Testing Numbers Considered Useful", BCP 82, RFC 3692, Jan. 2004.
- [RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC 6013, Jan. 2011.
- [Si11] Simpson, W., "TCP Cookie Transactions (TCPCT) Sockets Application Program Interface (API)", work in progress, draft-simpson-tcpct-api-04, Apr. 7, 2011.

## 10. Acknowledgments

This document was motivated by discussions on the IETF TCPM mailing list and by Wes Eddy's proposal [Ed11]. Yoshifumi Nishida, Pasi Sarolathi, and Michael Scharf provided detailed feedback.

This document was prepared using 2-Word-v2.0.template.dot.

## Authors' Addresses

Joe Touch  
USC/ISI  
4676 Admiralty Way  
Marina del Rey, CA 90292-6695 U.S.A.

Phone: +1 (310) 448-9151  
Email: touch@isi.edu



Internet Draft  
draft-ietf-tcpm-fastopen-10.txt  
Intended status: Experimental  
Expiration date: April, 2015

Y. Cheng  
J. Chu  
S. Radhakrishnan  
A. Jain  
Google  
September 29, 2014

## TCP Fast Open

### Abstract

This document describes an experimental TCP mechanism TCP Fast Open (TFO). TFO allows data to be carried in the SYN and SYN-ACK packets and consumed by the receiving end during the initial connection handshake, and saves up to one full round trip time (RTT) compared to the standard TCP, which requires a three-way handshake (3WHS) to complete before data can be exchanged. However TFO deviates from the standard TCP semantics since the data in the SYN could be replayed to an application in some rare circumstances. Applications should not use TFO unless they can tolerate this issue detailed in the Applicability section.

### Status of this Memo

Distribution of this memo is unlimited.

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

### Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Terminology . . . . .	3
2. Data In SYN . . . . .	3
2.1 Relaxing TCP Semantics on Duplicated SYNs . . . . .	4
2.2. SYNs with Spoofed IP Addresses . . . . .	4
3. Protocol Overview . . . . .	5
4. Protocol Details . . . . .	6
4.1. Fast Open Cookie . . . . .	6
4.1.1. Fast Open option . . . . .	7
4.1.2. Server Cookie Handling . . . . .	7
4.1.3. Client Cookie Handling . . . . .	8
4.1.3.1 Client Caching Negative Responses . . . . .	9
4.2. Fast Open Protocol . . . . .	9
4.2.1. Fast Open Cookie Request . . . . .	10
4.2.2. TCP Fast Open . . . . .	11
5. Security Considerations . . . . .	13
5.1. Resource Exhaustion Attack by SYN Flood with Valid Cookies . . . . .	13
5.1.1 Attacks from behind Shared Public IPs (NATs) . . . . .	14
5.2. Amplified Reflection Attack to Random Host . . . . .	14
6. TFO's Applicability . . . . .	15
6.1 Duplicate Data in SYNs . . . . .	15
6.2 Potential Performance Improvement . . . . .	16
6.3. Example: Web Clients and Servers . . . . .	16
6.3.1. HTTP Request Replay . . . . .	16
6.3.2. HTTP over TLS (HTTPS) . . . . .	16
6.3.3. Comparison with HTTP Persistent Connections . . . . .	17
6.3.4. Load Balancers and Server farms . . . . .	17
7. Open Areas for Experimentation . . . . .	17
7.1. Performance impact due to middle-boxes and NAT . . . . .	18
7.2. Impact on congestion control . . . . .	18
7.3. Cookie-less Fast Open . . . . .	18
8. Related Work . . . . .	19
8.1. T/TCP . . . . .	19
8.2. Common Defenses Against SYN Flood Attacks . . . . .	19
8.3. Speculative Connections by the Applications . . . . .	19

8.4. Fast Open Cookie in FIN . . . . .	19
8.5. TCP Cookie Transaction (TCPCT) . . . . .	20
9. IANA Considerations . . . . .	20
10. Acknowledgement . . . . .	20
11. References . . . . .	20
11.1. Normative References . . . . .	20
11.2. Informative References . . . . .	21
Appendix A. Example Socket API Changes to support TFO . . . . .	22
A.1 Active Open . . . . .	22
A.2 Passive Open . . . . .	23
Authors' Addresses . . . . .	23

## 1. Introduction

TCP Fast Open (TFO) is an experimental update to TCP that enables data to be exchanged safely during TCP's connection handshake. This document describes a design that enables applications to save a round trip while avoiding severe security ramifications. At the core of TFO is a security cookie used by the server side to authenticate a client initiating a TFO connection. This document covers the details of exchanging data during TCP's initial handshake, the protocol for TFO cookies, potential new security vulnerabilities and their mitigation, and the new socket API.

TFO is motivated by the performance needs of today's Web applications. Current TCP only permits data exchange after the 3-way handshake (3WHS)[RFC793], which adds one RTT to network latency. For short Web transfers this additional RTT is a significant portion of overall network latency, even when HTTP persistent connection is widely used. For example, the Chrome browser [Chrome] keeps TCP connections idle for up to 5 minutes but 35% of HTTP requests are made on new TCP connections [RCCJR11]. For such Web and Web-like applications placing data in the SYN can yield significant latency improvements. Next we describe how we resolve the challenges that arise upon doing so.

### 1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

TFO refers to TCP Fast Open. Client refers to the TCP's active open side and server refers to the TCP's passive open side.

## 2. Data In SYN

Standard TCP already allows data to be carried in SYN packets ([RFC793], section 3.4) but forbids the receiver from delivering it to the application until 3WSH is completed. This is because TCP's initial handshake serves to capture old or duplicate SYNs.

To enable applications exchange data in TCP handshake, TFO removes the constraint and allows data in SYN packets to be delivered to the application. This change of TCP semantic raises two issues discussed in the following subsections, making TFO unsuitable for certain applications.

Therefore TCP implementations MUST NOT use TFO by default, but only use TFO if requested explicitly by the application on a per service port basis. Applications need to evaluate TFO applicability described in Section 6 before using TFO.

## 2.1 Relaxing TCP Semantics on Duplicated SYNs

TFO allows data to be delivered to the application before the 3WSH is completed, thus opening itself to a data integrity issue in either of the two cases below:

a) the receiver host receives data in a duplicate SYN after it has forgotten it received the original SYN (e.g. due to a reboot);

b) the duplicate is received after the connection created by the original SYN has been closed and the close was initiated by the sender (so the receiver will not be protected by the 2MSL TIMEWAIT state).

The now obsoleted T/TCP [RFC1644] attempted to address these issues. It was not successful and not deployed due to various vulnerabilities as described in the Related Work section. Rather than trying to capture all dubious SYN packets to make TFO 100% compatible with TCP semantics, we made a design decision early on to accept old SYN packets with data, i.e., to restrict TFO use to a class of applications (Section 6) that are tolerant of duplicate SYN packets with data. We believe this is the right design trade-off balancing complexity with usefulness.

## 2.2. SYNs with Spoofed IP Addresses

Standard TCP suffers from the SYN flood attack [RFC4987] because SYN packets with spoofed source IP addresses can easily fill up a listener's small queue, causing a service port to be blocked completely until timeouts.

TFO goes one step further to allow server-side TCP to send up data to

the application layer before 3WSH is completed. This opens up serious new vulnerabilities. Applications serving ports that have TFO enabled may waste lots of CPU and memory resources processing the requests and producing the responses. If the response is much larger than the request, the attacker can further mount an amplified reflection attack against victims of choice beyond the TFO server itself.

Numerous mitigation techniques against regular SYN flood attacks exist and have been well documented [RFC4987]. Unfortunately none are applicable to TFO. We propose a server-supplied cookie to mitigate these new vulnerabilities in Section 3 and evaluate the effectiveness of the defense in Section 7.

### 3. Protocol Overview

The key component of TFO is the Fast Open Cookie (cookie), a message authentication code (MAC) tag generated by the server. The client requests a cookie in one regular TCP connection, then uses it for future TCP connections to exchange data during 3WSH:

Requesting a Fast Open Cookie:

1. The client sends a SYN with a Fast Open option with an empty cookie field to request a cookie.
2. The server generates a cookie and sends it through the Fast Open option of a SYN-ACK packet.
3. The client caches the cookie for future TCP Fast Open connections (see below).

Performing TCP Fast Open:

1. The client sends a SYN with data and the cookie in the Fast Open option.
2. The server validates the cookie:
  - a. If the cookie is valid, the server sends a SYN-ACK acknowledging both the SYN and the data. The server then delivers the data to the application.
  - b. Otherwise, the server drops the data and sends a SYN-ACK acknowledging only the SYN sequence number.
3. If the server accepts the data in the SYN packet, it may send the response data before the handshake finishes. The maximum amount is governed by the TCP's congestion control [RFC5681].

4. The client sends an ACK acknowledging the SYN and the server data. If the client's data is not acknowledged, the client retransmits the data in the ACK packet.
5. The rest of the connection proceeds like a normal TCP connection. The client can repeat many Fast Open operations once it acquires a cookie (until the cookie is expired by the server). Thus TFO is useful for applications that have temporal locality on client and server connections.

Requesting Fast Open Cookie in connection 1:

TCP A (Client)		TCP B(Server)
CLOSED		LISTEN
#1 SYN-SENT	----- <SYN, CookieOpt=NIL> ----->	SYN-RCVD
#2 ESTABLISHED (caches cookie C)	<----- <SYN, ACK, CookieOpt=C> -----	SYN-RCVD

Performing TCP Fast Open in connection 2:

TCP A (Client)		TCP B(Server)
CLOSED		LISTEN
#1 SYN-SENT	----- <SYN=x, CookieOpt=C, DATA_A> ----->	SYN-RCVD
#2 ESTABLISHED	<----- <SYN=y, ACK=x+len(DATA_A)+1> -----	SYN-RCVD
#3 ESTABLISHED	<----- <ACK=x+len(DATA_A)+1, DATA_B> -----	SYN-RCVD
#4 ESTABLISHED	----- <ACK=y+1>----->	ESTABLISHED
#5 ESTABLISHED	--- <ACK=y+len(DATA_B)+1>----->	ESTABLISHED

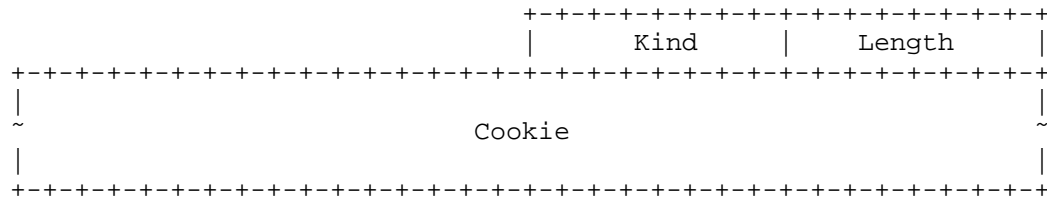
#### 4. Protocol Details

##### 4.1. Fast Open Cookie

The Fast Open Cookie is designed to mitigate new security vulnerabilities in order to enable data exchange during handshake. The cookie is a message authentication code tag generated by the server and is opaque to the client; the client simply caches the cookie and passes it back on subsequent SYN packets to open new connections. The server can expire the cookie at any time to enhance

security.

#### 4.1.1. Fast Open option



Kind                    1 byte: constant-TBD (to be assigned by IANA)  
 Length                1 byte: range 6 to 18 (bytes); limited by  
                               remaining space in the options field.  
                               The number MUST be even.  
 Cookie                0, or 4 to 16 bytes (Length - 2)

The Fast Open option is used to request or to send a Fast Open Cookie. When cookie is not present or empty, the option is used by the client to request a cookie from the server. When the cookie is present, the option is used to pass the cookie from the server to the client or from the client back to the server (to perform a Fast Open).

The minimum Cookie size is 4 bytes. Although the diagram shows a cookie aligned on 32-bit boundaries, alignment is not required. Options with invalid Length values or without SYN flag set MUST be ignored.

#### 4.1.2. Server Cookie Handling

The server is in charge of cookie generation and authentication. The cookie SHOULD be a message authentication code tag with the following properties. We use SHOULD because in some cases the cookie may be trivially generated as discussed in Section 7.3.

1. The cookie authenticates the client's (source) IP address of the SYN packet. The IP address may be an IPv4 or IPv6 address.
2. The cookie can only be generated by the server and can not be fabricated by any other parties including the client.
3. The generation and verification are fast relative to the rest of SYN and SYN-ACK processing.
4. A server may encode other information in the cookie, and accept more than one valid cookie per client at any given time. But this

is server implementation dependent and transparent to the client.

5. The cookie expires after a certain amount of time. The reason for cookie expiration is detailed in the "Security Consideration" section. This can be done by either periodically changing the server key used to generate cookies or including a timestamp when generating the cookie.

To gradually invalidate cookies over time, the server can implement key rotation to generate and verify cookies using multiple keys. This approach is useful for large-scale servers to retain Fast Open rolling key updates. We do not specify a particular mechanism because the implementation is server specific.

The server supports the cookie generation and verification operations:

- GetCookie(IP\_Address): returns a (new) cookie
- IsCookieValid(IP\_Address, Cookie): checks if the cookie is valid, i.e., it has not expired and it authenticates the client IP address.

Example Implementation: a simple implementation is to use AES\_128 to encrypt the IPv4 (with padding) or IPv6 address and truncate to 64 bits. The server can periodically update the key to expire the cookies. AES encryption on recent processors is fast and takes only a few hundred nanoseconds [RCCJR11].

If only one valid cookie is allowed per-IP and the server can regenerate the cookie independently, the best validation process is to simply regenerate a valid cookie and compare it against the incoming cookie. In that case if the incoming cookie fails the check, a valid cookie is readily available to be sent to the client.

#### 4.1.3. Client Cookie Handling

The client MUST cache cookies from servers for later Fast Open connections. For a multi-homed client, the cookies are dependent on the client and server IP addresses. Hence the client should cache at most one (most recently received) cookie per client and server IP addresses pair.

When caching cookies, we recommend that the client also cache the Maximum Segment Size (MSS) advertised by the server. The client can cache the MSS advertised by the server in order to determine the maximum amount of data that the client can fit in the SYN packet in

subsequent TFO connections. Caching the server MSS is useful because with Fast Open a client sends data in the SYN packet before the server announces its MSS in the SYN-ACK packet. If the client sends more data in the SYN packet than the server will accept, this will likely require the client to retransmit some or all of the data. Hence caching the server MSS can enhance performance.

Without a cached server MSS, the amount of data in the SYN packet is limited to the default MSS of 536 bytes for IPv4 [RFC1122] and 1240 bytes for IPv6 [RFC2460]. Even if the client complies with this limit when sending the SYN, it is known that an IPv4 receiver advertising an MSS less than 536 bytes can receive a segment larger than it is expecting.

If the cached MSS is larger than the typical size (1460 bytes for IPv4, or 1440 bytes for IPv6), then the excess data in the SYN packet may cause problems that offset the performance benefit of Fast Open. For example, the unusually large SYN may trigger IP fragmentation and may confuse firewalls or middleboxes, causing SYN retransmission and other side effects. Therefore the client MAY limit the cached MSS to 1460 bytes for IPv4 or 1440 for IPv6.

#### 4.1.3.1 Client Caching Negative Responses

The client MUST cache negative responses from the server in order to avoid potential connection failures. Negative responses include server not acknowledging the data in SYN, ICMP error messages, and most importantly no response (SYN/ACK) from the server at all, i.e., connection timeout. The last case is likely due to incompatible middle-boxes or firewall blocking the connection completely after it sees data in SYN. If the client does not react to these negative responses and continue to retry Fast Open, the client may never be able to connect to the specific server.

For any negative responses, the client SHOULD disable Fast Open on the specific path (the source and destination IP addresses and ports) at least temporarily. Since TFO is enabled on a per-service port basis but cookies are independent of service ports, the client's cache should include remote port numbers too.

#### 4.2. Fast Open Protocol

One predominant requirement of TFO is to be fully compatible with existing TCP implementations, both on the client and the server sides.

The server keeps two variables per listening socket (IP address & port):

FastOpenEnabled: default is off. It MUST be turned on explicitly by the application. When this flag is off, the server does not perform any TFO related operations and MUST ignore all cookie options.

PendingFastOpenRequests: tracks number of TFO connections in SYN-RCVD state. If this variable goes over a preset system limit, the server MUST disable TFO for all new connection requests until PendingFastOpenRequests drops below the system limit. This variable is used for defending some vulnerabilities discussed in the "Security Considerations" section.

The server keeps a FastOpened flag per connection to mark if a connection has successfully performed a TFO.

#### 4.2.1. Fast Open Cookie Request

Any client attempting TFO MUST first request a cookie from the server with the following steps:

1. The client sends a SYN packet with a Fast Open option with a length field of 0 (empty cookie field).
2. The server responds with a SYN-ACK based on the procedures in the "Server Cookie Handling" section. This SYN-ACK may contain a Fast Open option if the server currently supports TFO for this listener port.
3. If the SYN-ACK has a Fast Open option with a cookie, the client replaces the cookie and other information as described in the "Client Cookie Handling" section. Otherwise, if the SYN-ACK is first seen, i.e., not a (spurious) retransmission, the client MAY remove the server information from the cookie cache. If the SYN-ACK is a spurious retransmission, the client does nothing to the cookie cache for the reasons below.

The network or servers may drop the SYN or SYN-ACK packets with the new cookie options, which will cause SYN or SYN-ACK timeouts. We RECOMMEND both the client and the server to retransmit SYN and SYN-ACK without the cookie options on timeouts. This ensures the connections of cookie requests will go through and lowers the latency penalty (of dropped SYN/SYN-ACK packets). The obvious downside for maximum compatibility is that any regular SYN drop will fail the cookie (although one can argue the delay in the data transmission till after 3WSH is justified if the SYN drop is due to network congestion). The next section describes a heuristic to detect such drops when the client receives the SYN-ACK.

We also RECOMMEND the client to record the set of servers that failed to respond to cookie requests and only attempt another cookie request after certain period.

#### 4.2.2. TCP Fast Open

Once the client obtains the cookie from the target server, it can perform subsequent TFO connections until the cookie is expired by the server.

Client: Sending SYN

To open a TFO connection, the client MUST have obtained a cookie from the server:

1. Send a SYN packet.
  - a. If the SYN packet does not have enough option space for the Fast Open option, abort TFO and fall back to regular 3WSH.
  - b. Otherwise, include the Fast Open option with the cookie of the server. Include any data up to the cached server MSS or default 536 bytes.
2. Advance to SYN-SENT state and update SND.NXT to include the data accordingly.

To deal with network or servers dropping SYN packets with payload or unknown options, when the SYN timer fires, the client SHOULD retransmit a SYN packet without data and Fast Open options.

Server: Receiving SYN and responding with SYN-ACK

Upon receiving the SYN packet with Fast Open option:

1. Initialize and reset a local FastOpened flag. If FastOpenEnabled is false, go to step 5.
2. If PendingFastOpenRequests is over the system limit, go to step 5.
3. If IsCookieValid() in section 4.1.2 returns false, go to step 5.
4. Buffer the data and notify the application. Set FastOpened flag and increment PendingFastOpenRequests.
5. Send the SYN-ACK packet. The packet MAY include a Fast Open

Option. If FastOpened flag is set, the packet acknowledges the SYN and data sequence. Otherwise it acknowledges only the SYN sequence. The server MAY include data in the SYN-ACK packet if the response data is readily available. Some application may favor delaying the SYN-ACK, allowing the application to process the request in order to produce a response, but this is left up to the implementation.

6. Advance to the SYN-RCVD state. If the FastOpened flag is set, the server MUST follow [RFC5681] (based on [RFC3390]) to set the initial congestion window for sending more data packets.

If the SYN-ACK timer fires, the server SHOULD retransmit a SYN-ACK segment with neither data nor Fast Open options for compatibility reasons.

A special case is simultaneous open where the SYN receiver is a client in SYN-SENT state. The protocol remains the same because [RFC793] already supports both data in SYN and simultaneous open. But the client's socket may have data available to read before it's connected. This document does not cover the corresponding API change.

Client: Receiving SYN-ACK

The client SHOULD perform the following steps upon receiving the SYN-ACK:

1. If the SYN-ACK has a Fast Open option or MSS option or both, update the corresponding cookie and MSS information in the cookie cache.
2. Send an ACK packet. Set acknowledgment number to RCV.NXT and include the data after SND.UNA if data is available.
3. Advance to the ESTABLISHED state.

Note there is no latency penalty if the server does not acknowledge the data in the original SYN packet. The client SHOULD retransmit any unacknowledged data in the first ACK packet in step 2. The data exchange will start after the handshake like a regular TCP connection.

If the client has timed out and retransmitted only regular SYN packets, it can heuristically detect paths that intentionally drop SYN with Fast Open option or data. If the SYN-ACK acknowledges only the initial sequence and does not carry a Fast Open cookie option, presumably it is triggered by a retransmitted (regular) SYN and the

original SYN or the corresponding SYN-ACK was lost.

Server: Receiving ACK

Upon receiving an ACK acknowledging the SYN sequence, the server decrements PendingFastOpenRequests and advances to the ESTABLISHED state. No special handling is required further.

## 5. Security Considerations

The Fast Open cookie stops an attacker from trivially flooding spoofed SYN packets with data to burn server resources or to mount an amplified reflection attack on random hosts. The server can defend against spoofed SYN floods with invalid cookies using existing techniques [RFC4987]. We note that although generating bogus cookies is cost-free, the cost of validating the cookies, inherent to any authentication scheme, may be substantial compared to processing a regular SYN packet. We describe these new vulnerabilities of TFO and the countermeasures in detail below.

### 5.1. Resource Exhaustion Attack by SYN Flood with Valid Cookies

An attacker may still obtain cookies from some compromised hosts, then flood spoofed SYN with data and "valid" cookies (from these hosts or other vantage points). Like regular TCP handshakes, TFO is vulnerable to such an attack. But the potential damage can be much more severe. Besides causing temporary disruption to service ports under attack, it may exhaust server CPU and memory resources. Such an attack will show up on application server logs as an application level DoS from Bot-nets, triggering other defenses and alerts.

To protect the server it is important to limit the maximum number of total pending TFO connection requests, i.e., PendingFastOpenRequests (Section 4.2). When the limit is exceeded, the server temporarily disables TFO entirely as described in "Server Cookie Handling". Then subsequent TFO requests will be downgraded to regular connection requests, i.e., with the data dropped and only SYN acknowledged. This allows regular SYN flood defense techniques [RFC4987] like SYN-cookies to kick in and prevent further service disruption.

The main impact of SYN floods against the standard TCP stack is not directly from the floods themselves costing TCP processing overhead or host memory, but rather from the spoofed SYN packets filling up the often small listener's queue.

On the other hand, TFO SYN floods can cause damage directly if admitted without limit into the stack. The RST packets from the spoofed host will fuel rather than defeat the SYN floods as compared

to the non-TFO case, because the attacker can flood more SYNs with data to cost more data processing resources. For this reason, a TFO server needs to monitor the connections in SYN-RCVD being reset in addition to imposing a reasonable max queue length. Implementations may combine the two, e.g., by continuing to account for those connection requests that have just been reset against the listener's PendingFastOpenRequests until a timeout period has passed.

Limiting the maximum number of pending TFO connection requests does make it easy for an attacker to overflow the queue, causing TFO to be disabled. We argue that causing TFO to be disabled is unlikely to be of interest to attackers because the service will remain intact without TFO hence there is hardly any real damage.

#### 5.1.1 Attacks from behind Shared Public IPs (NATs)

An attacker behind a NAT can easily obtain valid cookies to launch the above attack to hurt other clients that share the path. [BRISCOE12] suggested that the server can extend cookie generation to include the TCP timestamp---GetCookie(IP\_Address, Timestamp)---and implement it by encrypting the concatenation of the two values to generate the cookie. The client stores both the cookie and its corresponding timestamp, and echoes both in the SYN. The server then implements IsCookieValid(IP\_Address, Timestamp, Cookie) by encrypting the IP and timestamp data and comparing it with the cookie value.

This enables the server to issue different cookies to clients that share the same IP address, hence can selectively discard those misused cookies from the attacker. However the attacker can simply repeat the attack with new cookies. The server would eventually need to throttle all requests from the IP address just like the current approach. Moreover this approach requires modifying [RFC1323] to send non-zero Timestamp Echo Reply in SYN, potentially causing firewall issues. Therefore we believe the benefit does not outweigh the drawbacks.

#### 5.2. Amplified Reflection Attack to Random Host

Limiting PendingFastOpenRequests with a system limit can be done without Fast Open Cookies and would protect the server from resource exhaustion. It would also limit how much damage an attacker can cause through an amplified reflection attack from that server. However, it would still be vulnerable to an amplified reflection attack from a large number of servers. An attacker can easily cause damage by tricking many servers to respond with data packets at once to any spoofed victim IP address of choice.

With the use of Fast Open Cookies, the attacker would first have to

steal a valid cookie from its target victim. This likely requires the attacker to compromise the victim host or network first. But in some cases it may be relatively easy.

The attacker here has little interest in mounting an attack on the victim host that has already been compromised. But it may be motivated to disrupt the victim's network. Since a stolen cookie is only valid for a single server, it has to steal valid cookies from a large number of servers and use them before they expire to cause sufficient damage without triggering the defense.

One can argue that if the attacker has compromised the target network or hosts, it could perform a similar but simpler attack by injecting bits directly. The degree of damage will be identical, but TFO-specific attack allows the attacker to remain anonymous and disguises the attack as from other servers.

For example with DHCP an attacker can obtain cookies when he (or the host he has compromised) owns a particular IP address by performing regular Fast Open to servers supporting TFO and collect valid cookies. The attacker then actively or passively releases his IP address. When the IP address is re-assigned to a victim, the attacker now owning a different IP address, floods spoofed Fast Open requests to perform an amplified reflection attack on the victim.

The best defense is for the server not to respond with data until handshake finishes. In this case the risk of amplification reflection attack is completely eliminated. But the potential latency saving from TFO may diminish if the server application produces responses earlier before the handshake completes.

## 6. TFO's Applicability

This section is to help applications considering TFO to evaluate TFO's benefits and drawbacks using the Web client and server applications as an example throughout. Applications here refer specifically to the process that writes data into the socket. For example a JavaScript process that sends data to the server. A proposed socket API change is in the Appendix.

### 6.1 Duplicate Data in SYNs

It is possible that using TFO results in the first data written to a socket to be delivered more than once to the application on the remote host (Section 2.1). This replay potential only applies to data in the SYN but not subsequent data exchanges.

Empirically [JIDKT07] showed the packet duplication on a Tier-1

network is rare. Since the replay only happens specifically when the SYN data packet is duplicated and also the duplicate arrives after the receiver has cleared the original SYN's connection state, the replay is thought to be uncommon in practice. Nevertheless a client that cannot handle receiving the same SYN data more than once MUST NOT enable TFO to send data in a SYN. Similarly a server that cannot accept receiving the same SYN data more than once MUST NOT enable TFO to receive data in a SYN. Further investigation is needed to judge about the probability of receiving duplicated SYN or SYN-ACK with data in non-Tier 1 networks.

## 6.2 Potential Performance Improvement

TFO is designed for latency-conscious applications that are sensitive to TCP's initial connection setup delay. To benefit from TFO, the first application data unit (e.g., an HTTP request) needs to be no more than TCP's maximum segment size (minus options used in SYN). Otherwise the remote server can only process the client's application data unit once the rest of it is delivered after the initial handshake, diminishing TFO's benefit.

To the extent possible, applications SHOULD reuse the connection to take advantage of TCP's built-in congestion control and reduce connection setup overhead. An application that employs too many short-lived connections will negatively impact network stability, as these connections often exit before TCP's congestion control algorithm takes effect.

## 6.3. Example: Web Clients and Servers

### 6.3.1. HTTP Request Replay

While TFO is motivated by Web applications, the browser should not use TFO to send requests in SYNs if those requests cannot tolerate replays. One example is POST requests without application-layer transaction protection (e.g., a unique identifier in the request header).

On the other hand, TFO is particularly useful for GET requests. GET requests replay could happen across striped TCP connections: after a server receives an HTTP request but before the ACKs of the requests reach the browser, the browser may timeout and retry the same request on another (possibly new) TCP connection. This differs from a TFO replay only in that the replay is initiated by the browser, not by the TCP stack.

### 6.3.2. HTTP over TLS (HTTPS)

For TLS over TCP, it is safe and useful to include TLS CLIENT\_HELLO in the SYN packet to save one RTT in TLS handshake. There is no concern about violating idem-potency. In particular it can be used alone with the speculative connection above.

#### 6.3.3. Comparison with HTTP Persistent Connections

Is TFO useful given the wide deployment of HTTP persistent connections? The short answer is yes. Studies [RCCJR11][AERG11] show that the average number of transactions per connection is between 2 and 4, based on large-scale measurements from both servers and clients. In these studies, the servers and clients both kept idle connections up to several minutes, well into "human think" time.

Keeping connections open and idle even longer risks a greater performance penalty. [HNESSK10][MQXMZ11] show that the majority of home routers and ISPs fail to meet the 124-minute idle timeout mandated in [RFC5382]. In [MQXMZ11], 35% of mobile ISPs silently timeout idle connections within 30 minutes. End hosts, unaware of silent middle-box timeouts, suffer multi-minute TCP timeouts upon using those long-idle connections.

To circumvent this problem, some applications send frequent TCP keep-alive probes. However, this technique drains power on mobile devices [MQXMZ11]. In fact, power has become such a prominent issue in modern LTE devices that mobile browsers close HTTP connections within seconds or even immediately [SOUDERS11].

[RCCJR11] studied Chrome browser [Chrome] performance based on 28 days of global statistics. The Chrome browser keeps idle HTTP persistent connections for 5 to 10 minutes. However the average number of the transactions per connection is only 3.3 and TCP 3WSH accounts for up to 25% of the HTTP transaction network latency. The authors estimated that TFO improves page load time by 10% to 40% on selected popular Web sites.

#### 6.3.4. Load Balancers and Server farms

Servers behind a load balancers that accept connection requests to the same server IP address should use the same key such that they generate identical Fast Open Cookies for a particular client IP address. Otherwise a client may get different cookies across connections; its Fast Open attempts would fall back to regular 3WSH.

### 7. Open Areas for Experimentation

We now outline some areas that need experimentation in the Internet and under different network scenarios. These experiments should help

the community evaluate Fast Open benefits and risks towards further standardization and implementation of Fast Open and its related protocols.

#### 7.1. Performance impact due to middle-boxes and NAT

[MAF04] found that some middle-boxes and end-hosts may drop packets with unknown TCP options. Studies [LANGLEY06, HNRGHT11] both found that 6% of the probed paths on the Internet drop SYN packets with data or with unknown TCP options. The TFO protocol deals with this problem by falling back to regular TCP handshake and re-transmitting SYN without data or cookie options after the initial SYN timeout. Moreover the implementation is recommended to negatively cache such incidents to avoid recurring timeouts. Further study is required to evaluate the performance impact of these drop behaviors.

Another interesting study is the loss of TFO performance benefit behind certain carrier-grade NAT. Typically hosts behind a NAT sharing the same IP address will get the same cookie for the same server. This will not prevent TFO from working. But on some carrier-grade NAT configurations where every new TCP connection from the same physical host uses a different public IP address, TFO does not provide latency benefits. However, there is no performance penalty either, as described in Section "Client: Receiving SYN-ACK".

#### 7.2. Impact on congestion control

Although TFO does not directly change the congestion control, there are subtle cases that it may. When SYN-ACK times out, regular TCP reduces the initial congestion window before sending any data [RFC5681]. However in TFO the server may have already sent up to an initial window of data.

If the server serves mostly short connections then the losses of SYN-ACKs are not as effective as regular TCP on reducing the congestion window. This could result in an unstable network condition. The connections that experience losses may attempt again and add more load under congestion. A potential solution is to temporarily disable Fast Open if the server observes many SYN-ACK or data losses during the handshake across connections. Further experimentation regarding the congestion control impact will be useful.

#### 7.3. Cookie-less Fast Open

The cookie mechanism mitigates resource exhaustion and amplification attacks. However cookies are not necessary if the server has application-level protection or is immune to these attacks. For example a Web server that only replies with a simple HTTP redirect

response that fits in the SYN-ACK packet may not care about resource exhaustion.

For such applications the server may choose to generate a trivial or even a zero-length cookie to improve performance by avoiding the cookie generation and verification. If the server believes it's under a DoS attack through other defense mechanisms, it can switch to regular Fast Open for listener sockets.

## 8. Related Work

### 8.1. T/TCP

TCP Extensions for Transactions [RFC1644] attempted to bypass the three-way handshake, among other things, hence shared the same goal but also the same set of issues as TFO. It focused most of its effort battling old or duplicate SYNs, but paid no attention to security vulnerabilities it introduced when bypassing 3WS [PHRACK98].

As stated earlier, we take a practical approach to focus TFO on the security aspect, while allowing old, duplicate SYN packets with data after recognizing that 100% TCP semantics is likely infeasible. We believe this approach strikes the right tradeoff, and makes TFO much simpler and more appealing to TCP implementers and users.

### 8.2. Common Defenses Against SYN Flood Attacks

[RFC4987] studies on mitigating attacks from regular SYN flood, i.e., SYN without data. But from the stateless SYN-cookies to the stateful SYN Cache, none can preserve data sent with SYN safely while still providing an effective defense.

The best defense may be to simply disable TFO when a host is suspected to be under a SYN flood attack, e.g., the SYN backlog is filled. Once TFO is disabled, normal SYN flood defenses can be applied. The "Security Consideration" section contains a thorough discussion on this topic.

### 8.3. Speculative Connections by the Applications

Some Web browsers maintain a history of the domains for frequently visited web pages. The browsers then speculatively pre-open TCP connections to these domains before the user initiates any requests for them [BELSHE11]. While this technique also saves the handshake latency, it wastes server and network resources by initiating and maintaining idle connections.

### 8.4. Fast Open Cookie in FIN

An alternate proposal is to request a TFO cookie in the FIN instead, since FIN-drop by incompatible middle-boxes does not affect latency. However paths that block SYN cookies may be more likely to drop a later SYN packet with data, and many applications close a connection with RST instead anyway.

Although cookie-in-FIN may not improve robustness, it would give clients using a single connection a latency advantage over clients opening multiple parallel connections. If experiments with TFO find that it leads to increased connection-sharding, cookie-in-FIN may prove to be a useful alternative.

#### 8.5. TCP Cookie Transaction (TCPCT)

TCPCT [RFC6013] eliminates server state during initial handshake and defends spoofing DoS attacks. Like TFO, TCPCT allows SYN and SYN-ACK packets to carry data. But the server can only send up to MSS bytes of data during the handshake instead of the initial congestion window unlike TFO. Therefore the latency of applications such as Web may be worse than with TFO.

#### 9. IANA Considerations

IANA is requested to allocate one value from the TCP Option Kind Numbers: The constant-TBD in Section 4.1.1 has to be replaced with the newly assigned value. The length of the new TCP option Kind is variable and the Meaning should be set to "TCP Fast Open Cookie". Early implementation before the IANA allocation SHOULD follow [RFC6994] and use experimental option 254 and magic number 0xF989 (16 bits), then migrate to the new option after the allocation accordingly.

#### 10. Acknowledgement

We thank Bob Briscoe, Michael Scharf, Gorrry Fairhurst, Rick Jones, Roberto Peon, William Chan, Adam Langley, Neal Cardwell, Eric Dumazet, and Matt Mathis for their feedbacks. We especially thank Barath Raghavan for his contribution on the security design of Fast Open and proofreading this draft numerous times.

#### 11. References

##### 11.1. Normative References

- [RFC793] Postel, J. "Transmission Control Protocol", RFC 793, September 1981.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts -

Communication Layers", STD 3, RFC 1122, October 1989.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5382] S. Guha, Ed., Biswas, K., Ford B., Sivakumar S., Srisuresh, P., "NAT Behavioral Requirements for TCP", RFC 5382
- [RFC5681] Allman, M., Paxson, V. and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009
- [RFC6994] Touch, Joe, "Shared Use of Experimental TCP Options", RFC6994, August 2013.
- [RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.

#### 11.2. Informative References

- [AERG11] Al-Fares, M., Elmeleegy, K., Reed, B., Gashinsky, I., "Overclocking the Yahoo! CDN for Faster Web Page Loads". In Proceedings of Internet Measurement Conference, November 2011.
- [HNESSK10] Haetoeinen, S., Nyrhinen, A., Eggert, L., Strowes, S., Sarolahti, P., Kojo., M., "An Experimental Study of Home Gateway Characteristics". In Proceedings of Internet Measurement Conference. October 2010
- [HNRGHT11] Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., Tokuda, H., "Is it Still Possible to Extend TCP?". In Proceedings of Internet Measurement Conference. November 2011.
- [LANGLEY06] Langley, A, "Probing the viability of TCP extensions", URL <http://www.imperialviolet.org/binary/ecntest.pdf>
- [MAF04] Medina, A., Allman, M., and S. Floyd, "Measuring Interactions Between Transport Protocols and Middleboxes". In Proceedings of Internet Measurement Conference, October 2004.
- [MQXMZ11] Wang, Z., Qian, Z., Xu, Q., Mao, Z., Zhang, M., "An Untold Story of Middleboxes in Cellular Networks". In Proceedings of SIGCOMM. August 2011.
- [PHRACK98] "T/TCP vulnerabilities", Phrack Magazine, Volume 8, Issue 53 artical 6. July 8, 1998. URL

<http://www.phrack.com/issues.html?issue=53&id=6>

- [RCCJR11] Radhakrishnan, S., Cheng, Y., Chu, J., Jain, A., Raghavan, B., "TCP Fast Open". In Proceedings of 7th ACM CoNEXT Conference, December 2011.
- [RFC1323] Jacobson, V., Braden, R., Borman, D., "TCP Extensions for High Performance", RFC 1323, May 1992.
- [RFC1644] Braden, R., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, July 1994.
- [RFC2460] Deering, S., Hinden, R., "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.
- [RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC6013, January 2011.
- [SOUDERS11] Souders, S., "Making A Mobile Connection". <http://www.stevesouders.com/blog/2011/09/21/making-a-mobile-connection/>
- [BRISCOE12] Briscoe, B., "Some ideas building on draft-ietf-tcpm-fastopen-01", tcpm list, <http://www.ietf.org/mail-archive/web/tcpm/current/msg07192.html>
- [BELSHE11] Belshe, M., "The era of browser preconnect.", <http://www.belshe.com/2011/02/10/the-era-of-browser-preconnect/>
- [JIDKT07] Jaiswal, S., Iannaccone, G., Diot, C., Kurose, J., Towsley, D., "Measurement and classification of out-of-sequence packets in a tier-1 IP backbone". IEEE/ACM Transactions on Networking (TON), 15(1), 54-66.
- [Chrome] Chrome. <https://www.google.com/intl/en-US/chrome/browser/>

## Appendix A. Example Socket API Changes to support TFO

### A.1 Active Open

The active open side involves changing or replacing the `connect()` call, which does not take a user data buffer argument. We recommend replacing `connect()` call to minimize API changes and hence

applications to reduce the deployment hurdle.

One solution implemented in Linux 3.7 is introducing a new flag `MSG_FASTOPEN` for `sendto()` or `sendmsg()`. `MSG_FASTOPEN` marks the attempt to send data in SYN like a combination of `connect()` and `sendto()`, by performing an implicit `connect()` operation. It blocks until the handshake has completed and the data is buffered.

For non-blocking socket it returns the number of bytes buffered and sent in the SYN packet. If the cookie is not available locally, it returns `-1` with `errno EINPROGRESS`, and sends a SYN with TFO cookie request automatically. The caller needs to write the data again when the socket is connected. On errors, it returns the same `errno` as `connect()` if the handshake fails.

An implementation may prefer not to change the `sendmsg()` because TFO is a TCP specific feature. A solution is to add a new socket option `TCP_FASTOPEN` for TCP sockets. When the option is enabled before a `connect` operation, `sendmsg()` or `sendto()` will perform Fast Open operation similar to the `MSG_FASTOPEN` flag described above. This approach however requires an extra `setsockopt()` system call.

## A.2 Passive Open

The passive open side change is simpler compared to active open side. The application only needs to enable the reception of Fast Open requests via a new `TCP_FASTOPEN` `setsockopt()` socket option before `listen()`.

The option enables Fast Open on the listener socket. The option value specifies the `PendingFastOpenRequests` threshold, i.e., the maximum length of pending SYNs with data payload. Once enabled, the TCP implementation will respond with TFO cookies per request.

Traditionally `accept()` returns only after a socket is connected. But for a Fast Open connection, `accept()` returns upon receiving a SYN with a valid Fast Open cookie and data, and the data is available to be read through, e.g., `recvmsg()`, `read()`.

## Authors' Addresses

Yuchung Cheng  
Google, Inc.  
1600 Amphitheatre Parkway  
Mountain View, CA 94043, USA  
EMail: ycheng@google.com

Jerry Chu

Google, Inc.  
1600 Amphitheatre Parkway  
Mountain View, CA 94043, USA  
EMail: hkchu@google.com

Sivasankar Radhakrishnan  
Department of Computer Science and Engineering  
University of California, San Diego  
9500 Gilman Dr  
La Jolla, CA 92093-0404  
EMail: sivasankar@cs.ucsd.edu

Arvind Jain  
Google, Inc.  
1600 Amphitheatre Parkway  
Mountain View, CA 94043, USA  
EMail: arvind@google.com

Internet Draft  
draft-ietf-tcpm-initcwnd-08.txt  
Intended status: Experimental

Expiration date: August 2013

J. Chu  
N. Dukkipati  
Y. Cheng  
M. Mathis  
Google, Inc.  
February 22, 2013

## Increasing TCP's Initial Window

### Status of this Memo

Distribution of this memo is unlimited.

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on August, 2013.

### Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Abstract

This document proposes an experiment to increase the permitted TCP initial window (IW) from between 2 and 4 segments, as specified in RFC 3390, to 10 segments, with a fallback to the existing recommendation when performance issues are detected. It discusses the motivation behind the increase, the advantages and disadvantages of the higher initial window, and presents results from several large scale experiments showing that the higher initial window improves the overall performance of many web services without resulting in a congestion collapse. The document closes with a discussion of usage and deployment for further experimental purpose recommended by the IETF TCP Maintenance and Minor Extensions (TCPM) working group.

## Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

## Table of Contents

1. Introduction . . . . .	3
2. TCP Modification . . . . .	4
3. Implementation Issues . . . . .	5
4. Background . . . . .	6
5. Advantages of Larger Initial Windows . . . . .	7
5.1 Reducing Latency . . . . .	7
5.2 Keeping up with the growth of web object size . . . . .	8
5.3 Recovering faster from loss on under-utilized or wireless links . . . . .	8
7. Disadvantages of Larger Initial Windows for the Network . . . . .	9
8. Mitigation of Negative Impact . . . . .	10
9. Interactions with the Retransmission Timer . . . . .	10
10. Experimental Results From Large Scale Cluster Tests . . . . .	10
10.1 The benefits . . . . .	11
10.2 The cost . . . . .	11
11. Other Studies . . . . .	12
12. Usage and Deployment Recommendations . . . . .	13
13. Related Proposals . . . . .	14
14. Security Considerations . . . . .	14
15. Conclusion . . . . .	15
16. IANA Considerations . . . . .	15
17. Acknowledgments . . . . .	15
Normative References . . . . .	16
Informative References . . . . .	16
Appendix A – List of Concerns and Corresponding Test Results . . . . .	20

Author's Addresses . . . . .	23
Acknowledgment . . . . .	23

## 1. Introduction

This document proposes to raise the upper bound on TCP's initial window (IW) to 10 segments (maximum 14600B). It is patterned after and borrows heavily from RFC 3390 [RFC3390] and earlier work in this area. Due to lingering concerns about possible side effects to other flows sharing the same network bottleneck, some of the recommendations are conditional on additional monitoring and evaluation.

The primary argument in favor of raising IW follows from the evolving scale of the Internet. Ten segments are likely to fit into queue space available at any broadband access link, even when there are a reasonable number of concurrent connections.

Lower speed links can be treated with environment specific configurations, such that they can be protected from being overwhelmed by large initial window bursts without imposing a suboptimal initial window on the rest of the Internet.

This document reviews the advantages and disadvantages of using a larger initial window, and includes summaries of several large scale experiments showing that an initial window of 10 segments provides benefits across the board for a variety of BW, RTT, and BDP classes. These results show significant benefits for increasing IW for users at much smaller data rates than had been previously anticipated. However, at initial windows larger than 10, the results are mixed. We believe that these mixed results are not intrinsic, but are the consequence of various implementation artifacts, including overly aggressive applications employing many simultaneous connections.

We recommend that all TCP implementations have a settable TCP IW parameter as long as there is a reasonable effort to monitor for possible interactions with other Internet applications and services as described in Section 12. Furthermore, Section 10 details why 10 segments may be an appropriate value, and while that value may continue to rise in the future, this document does not include any supporting evidence for values of IW larger than 10.

In addition, we introduce a minor revision to RFC 3390 and RFC 5681 [RFC5681] to eliminate resetting the initial window when the SYN or SYN/ACK is lost.

The document closes with a discussion of the consensus from the TCPM

working group on the near-term usage and deployment of IW10 in the Internet.

A complementary set of slides for this proposal can be found at [CD10].

## 2. TCP Modification

This document proposes an increase in the permitted upper bound for TCP's initial window (IW) to 10 segments depending on the MSS. This increase is optional: a TCP MAY start with an initial window that is smaller than 10 segments.

More precisely, the upper bound for the initial window will be

$$\min(10 * \text{MSS}, \max(2 * \text{MSS}, 14600)) \quad (1)$$

This upper bound for the initial window size represents a change from RFC 3390 [RFC3390], which specified that the congestion window be initialized between 2 and 4 segments depending on the MSS.

This change applies to the initial window of the connection in the first round trip time (RTT) of data transmission during or following the TCP three-way handshake. Neither the SYN/ACK nor its acknowledgment (ACK) in the three-way handshake should increase the initial window size.

Note that all the test results described in this document were based on the regular Ethernet MTU of 1500 bytes. Future study of the effect of a different MTU may be needed to fully validate (1) above.

Furthermore, RFC 3390 and RFC 5681 [RFC5681] state that

"If the SYN or SYN/ACK is lost, the initial window used by a sender after a correctly transmitted SYN MUST be one segment consisting of MSS bytes."

The proposed change to reduce the default RTO to 1 second [RFC6298] increases the chance for spurious SYN or SYN/ACK retransmission, thus unnecessarily penalizing connections with RTT > 1 second if their initial window is reduced to 1 segment. For this reason, it is RECOMMENDED that implementations refrain from resetting the initial window to 1 segment, unless either there have been more than one SYN or SYN/ACK retransmissions, or true loss detection has been made.

TCP implementations use slow start in as many as three different ways: (1) to start a new connection (the initial window); (2) to restart transmission after a long idle period (the restart window);

and (3) to restart transmission after a retransmit timeout (the loss window). The change specified in this document affects the value of the initial window. Optionally, a TCP MAY set the restart window to the minimum of the value used for the initial window and the current value of cwnd (in other words, using a larger value for the restart window should never increase the size of cwnd). These changes do NOT change the loss window, which must remain 1 segment of MSS bytes (to permit the lowest possible window size in the case of severe congestion).

Furthermore, to limit any negative effect that a larger initial window may have on links with limited bandwidth or buffer space, implementations SHOULD fall back to RFC 3390 for the restart window (RW) if any packet loss is detected during either the initial window, or a restart window, and more than 4KB of data is sent. Implementations must also follow RFC6298 [RFC6298] in order to avoid spurious RTO as described in section 9 later.

### 3. Implementation Issues

HTTP 1.1 specification allows only two simultaneous connections per domain, while web browsers open more simultaneous TCP connections [Ste08], partly to circumvent the small initial window in order to speed up the loading of web pages as described above.

When web browsers open simultaneous TCP connections to the same destination, they are working against TCP's congestion control mechanisms [FF99]. Combining this behavior with larger initial windows further increases the burstiness and unfairness to other traffic in the network. If a larger initial window causes harm to any other flows then local application tuning will reveal that fewer concurrent connections yields better performance for some users. Any content provider deploying IW10 in conjunction with content distributed across multiple domains is explicitly encouraged to perform measurement experiments to detect such problems, and to consider reducing the number of concurrent connections used to retrieve their content.

Some implementations advertise small initial receive window (Table 2 in [Duk10]), effectively limiting how much window a remote host may use. In order to realize the full benefit of the large initial window, implementations are encouraged to advertise an initial receive window of at least 10 segments, except for the circumstances where a larger initial window is deemed harmful. (See the Mitigation section below.)

TCP SACK option ([RFC2018]) was thought to be required in order for the larger initial window to perform well. But measurements from both

a testbed and live tests showed that IW=10 without the SACK option outperforms IW=3 with the SACK option [CW10].

#### 4. Background

TCP congestion window was introduced as part of the congestion control algorithm by Van Jacobson in 1988 [Jac88]. The initial value of one segment was used as the starting point for newly established connections to probe the available bandwidth on the network.

Today's Internet is dominated by web traffic running on top of short-lived TCP connections [IOR2009]. The relatively small initial window has become a limiting factor for the performance of many web applications.

The global Internet has continued to grow, both in speed and penetration. According to the latest report from Akamai [AKAM10], the global broadband (> 2Mbps) adoption has surpassed 50%, propelling the average connection speed to reach 1.7Mbps, while the narrowband (< 256Kbps) usage has dropped to 5%. In contrast, TCP's initial window has remained 4KB for a decade [RFC2414], corresponding to a bandwidth utilization of less than 200Kbps per connection, assuming an RTT of 200ms.

A large proportion of flows on the Internet are short web transactions over TCP, and complete before exiting TCP slow start. Speeding up the TCP flow startup phase, including circumventing the initial window limit, has been an area of active research [RFC6077, Sch08]. Numerous proposals exist [LAJW07, RFC4782, PRAKS02, PK98]. Some require router support [RFC4782, PK98], hence are not practical for the public Internet. Others suggested bold, but often radical ideas, likely requiring more years of research before standardization and deployment.

In the mean time, applications have responded to TCP's "slow" start. Web sites use multiple sub-domains [Bell0] to circumvent HTTP 1.1 regulation on two connections per physical host [RFC2616]. As of today, major web browsers open multiple connections to the same site (up to six connections per domain [Ste08] and the number is growing). This trend is to remedy HTTP serialized download to achieve parallelism and higher performance. But it also implies today most access links are severely under-utilized, hence having multiple TCP connections improves performance most of the time. While raising the initial congestion window may cause congestion for certain users using these browsers, we argue that the browsers and other application need to respect HTTP 1.1 regulation and stop increasing number of simultaneous TCP connections. We believe a modest increase of the initial window will help to stop this trend, and provide the

best interim solution to improve overall user performance, and reduce the server, client, and network load.

Note that persistent connections and pipelining are designed to address some of the above issues with HTTP [RFC2616]. Their presence does not diminish the need for a larger initial window. E.g., data from the Chrome browser show that 35% of HTTP requests are made on new TCP connections. Our test data also shows significant latency reduction with the large initial window even in conjunction with these two HTTP features ([Duk10]).

Also note that packet pacing has been suggested as a possible mechanism to avoid large bursts and their associated harm [VH97]. Pacing is not required in this proposal due to a strong preference for a simple solution. We suspect for packet bursts of a moderate size, packet pacing will not be necessary. This seems to be confirmed by our test results.

More discussion of the increase in initial window, including the choice of 10 segments can be found in [Duk10, CD10].

## 5. Advantages of Larger Initial Windows

### 5.1 Reducing Latency

An increase of the initial window from 3 segments to 10 segments reduces the total transfer time for data sets greater than 4KB by up to 4 round trips.

The table below compares the number of round trips between IW=3 and IW=10 for different transfer sizes, assuming infinite bandwidth, no packet loss, and the standard delayed acks with large delayed-ACK timer.

total segments	IW=3	IW=10
3	1	1
6	2	1
10	3	1
12	3	2
21	4	2
25	5	2
33	5	3
46	6	3
51	6	4
78	7	4
79	8	4

	120		8		5	
	127		9		5	
-----						

For example, with the larger initial window, a transfer of 32 segments of data will require only two rather than five round trips to complete.

## 5.2 Keeping up with the growth of web object size

RFC 3390 stated that the main motivation for increasing the initial window to 4KB was to speed up connections that only transmit a small amount of data, e.g., email and web. The majority of transfers back then were less than 4KB, and could be completed in a single RTT [All100].

Since RFC 3390 was published, web objects have gotten significantly larger [Chu09, RJ10]. Today only a small percentage of web objects (e.g., 10% of Google's search responses) can fit in the 4KB initial window. The average HTTP response size of gmail.com, a highly scripted web-site, is 8KB (Figure 1. in [Duk10]). The average web page, including all static and dynamic scripted web objects on the page, has seen even greater growth in size [RJ10]. HTTP pipelining [RFC2616] and new web transport protocols such as SPDY [SPDY] allow multiple web objects to be sent in a single transaction, potentially benefiting from an even larger initial window in order to transfer an entire web page in a small number of round trips.

## 5.3 Recovering faster from loss on under-utilized or wireless links

A greater-than-3-segment initial window increases the chance to recover packet loss through Fast Retransmit rather than the lengthy initial RTO [RFC5681]. This is because the fast retransmit algorithm requires three duplicate ACKs as an indication that a segment has been lost rather than reordered. While newer loss recovery techniques such as Limited Transmit [RFC3042] and Early Retransmit [RFC5827] have been proposed to help speeding up loss recovery from a smaller window, both algorithms can still benefit from the larger initial window because of a better chance to receive more ACKs to react upon.

## 6. Disadvantages of Larger Initial Windows for the Individual Connection

The larger bursts from an increase in the initial window may cause buffer overrun and packet drop in routers with small buffers, or routers experiencing congestion. This could result in unnecessary retransmit timeouts. For a large-window connection that is able to recover without a retransmit timeout, this could result in an unnecessarily-early transition from the slow-start to the congestion-

avoidance phase of the window increase algorithm.

Premature segment drops are unlikely to occur in uncongested networks with sufficient buffering, or in moderately-congested networks where the congested router uses active queue management (such as Random Early Detection [FJ93, RFC2309, RFC3150]).

Insufficient buffering is more likely to exist in the access routers connecting slower links. A recent study of access router buffer size [DGHS07] reveals the majority of access routers provision enough buffer for 130ms or longer, sufficient to cover a burst of more than 10 packets at 1Mbps speed, but possibly not sufficient for browsers opening simultaneous connections.

A testbed study [CW10] on the effect of the larger initial window with five simultaneously opened connections revealed that, even with limited buffer size on slow links, IW=10 still reduced the total latency of web transactions, although at the cost of higher packet drop rates as compared to IW=3.

Some TCP connections will receive better performance with the larger initial window even if the burstiness of the initial window results in premature segment drops. This will be true if (1) the TCP connection recovers from the segment drop without a retransmit timeout, and (2) the TCP connection is ultimately limited to a small congestion window by either network congestion or by the receiver's advertised window.

## 7. Disadvantages of Larger Initial Windows for the Network

An increase in the initial window may increase congestion in a network. However, since the increase is one-time only (at the beginning of a connection), and the rest of TCP's congestion backoff mechanism remains in place, it's unlikely the increase by itself will render a network in a persistent state of congestion, or even congestion collapse. This seems to have been confirmed by the large scale web experiments described later.

It should be noted that the above may not hold if applications open a large number of simultaneous connections.

Until this proposal is widely deployed, a fairness issue may exist between flows adopting a larger initial window vs flows that are RFC3390-compliant. Although no severe unfairness has been detected on all the known tests so far, further study on this topic may be warranted.

Some of the discussions from RFC 3390 are still valid for IW=10.

Moreover, it is worth noting that although TCP NewReno increases the chance of duplicate segments when trying to recover multiple packet losses from a large window, the wide support of TCP Selective Acknowledgment (SACK) option [RFC2018] in all major OSes today should keep the volume of duplicate segments in check.

Recent measurements [Get11] provide evidence of extremely large queues (in the order of one second or more) at access networks of the Internet. While a significant part of the buffer bloat is contributed by large downloads/uploads such as video files, emails with large attachments, backups and download of movies to disk, some of the problem is also caused by Web browsing of image heavy sites [Get11]. This queuing delay is generally considered harmful for responsiveness of latency sensitive traffic such as DNS queries, ARP, DHCP, VoIP and Gaming. IW=10 can exacerbate this problem when doing short downloads such as Web browsing [Get11-1]. The mitigations proposed for the broader problem of buffer bloating are also applicable in this case, such as the use of ECN, AQM schemes [CoDel] and traffic classification (QoS).

#### 8. Mitigation of Negative Impact

Much of the negative impact from an increase in the initial window is likely to be felt by users behind slow links with limited buffers. The negative impact can be mitigated by hosts directly connected to a low-speed link advertising a smaller initial receive window than 10 segments. This can be achieved either through manual configuration by the users, or through the host stack auto-detecting the low bandwidth links.

Additional suggestions to improve the end-to-end performance of slow links can be found in RFC 3150 [RFC3150].

#### 9. Interactions with the Retransmission Timer

A large initial window increases the chance of spurious RTO on a low-bandwidth path because the packet transmission time will dominate the round-trip time. To minimize spurious retransmissions, implementations MUST follow RFC 6298 [RFC6298] to restart the retransmission timer with the current value of RTO for each ACK received that acknowledges new data.

For a more detailed discussion see RFC3390, section 6.

#### 10. Experimental Results From Large Scale Cluster Tests

In this section we summarize our findings from large scale Internet experiments with an initial window of 10 segments, conducted via

Google's front-end infrastructure serving a diverse set of applications. We present results from two data centers, each chosen because of the specific characteristics of subnets served: AvgDC has connection bandwidths closer to the worldwide average reported in [AKAM10], with a median connection speed of about 1.7Mbps; SlowDC has a larger proportion of traffic from slow bandwidth subnets with nearly 20% of traffic from connections below 100Kbps, and a third below 256Kbps.

Guided by measurements data, we answer two key questions: what is the latency benefit when TCP connections start with a higher initial window, and on the flip side, what is the cost?

### 10.1 The benefits

The average web search latency improvement over all responses in AvgDC is 11.7% (68 ms) and 8.7% (72 ms) in SlowDC. We further analyzed the data based on traffic characteristics and subnet properties such as bandwidth (BW), round-trip time (RTT), and bandwidth-delay product (BDP). The average response latency improved across the board for a variety of subnets with the largest benefits of over 20% from high RTT and high BDP networks, wherein most responses can fit within the pipe. Correspondingly, responses from low RTT paths experienced the smallest improvements of about 5%.

Contrary to what we expected, responses from low bandwidth subnets experienced the best latency improvements (between 10-20%) in the buckets 0-56Kbps and 56-256Kbps buckets. We speculate low BW networks observe improved latency for two plausible reasons: 1) fewer slow-start rounds: unlike many large BW networks, low BW subnets with dial-up modems have inherently large RTTs; and 2) faster loss recovery: an initial window larger than 3 segments increases the chances of a lost packet to be recovered through Fast Retransmit as opposed to a lengthy RTO.

Responses of different sizes benefited to varying degrees; those larger than 3 segments naturally demonstrated larger improvements, because they finished in fewer rounds in slow start as compared to the baseline. In our experiments, response sizes  $\leq 3$  segments also demonstrated small latency benefits.

To find out how individual subnets performed, we analyzed average latency at a /24 subnet level (an approximation to a user base offered similar set of services by a common ISP). We find even at the subnet granularity, latency improved at all quantiles ranging from 5-11%.

### 10.2 The cost

To quantify the cost of raising the initial window, we analyzed the data specifically for subnets with low bandwidth and BDP, retransmission rates for different kinds of applications, as well as latency for applications operating with multiple concurrent TCP connections. From our measurements we found no evidence of a negative latency impacts that correlate to BW or BDP alone, but in fact both kinds of subnets demonstrated latency improvements across averages and quantiles.

As expected, the retransmission rate increased modestly when operating with larger initial congestion window. The overall increase in AvgDC is 0.3% (from 1.98% to 2.29%) and in SlowDC is 0.7% (from 3.54% to 4.21%). In our investigation, with the exception of one application, the larger window resulted in a retransmission increase of < 0.5% for services in the AvgDC. The exception is the Maps application that operates with multiple concurrent TCP connections, which increased its retransmission rate by 0.9% in AvgDC and 1.85% in SlowDC (from 3.94% to 5.79%).

In our experiments, the percentage of traffic experiencing retransmissions did not increase significantly. E.g. 90% of web search and maps experienced zero retransmission in SlowDC (percentages are higher for AvgDC); a break up of retransmissions by percentiles indicate that most increases come from portion of traffic already experiencing retransmissions in the baseline with initial window of 3 segments.

Traffic patterns from applications using multiple concurrent TCP connections all operating with a large initial window represent one of the worst case scenarios where latency can be adversely impacted due to bottleneck buffer overflow. Our investigation shows that such a traffic pattern has not been a problem in AvgDC, where all these applications, specifically maps and image thumbnails, demonstrated improved latencies varying from 2-20%. In the case of SlowDC, while these applications continued showing a latency improvement in the mean, their latencies in higher quantiles (96 and above for maps) indicated instances where latency with larger window is worse than the baseline, e.g. the 99% latency for maps has increased by 2.3% (80ms) when compared to the baseline. There is no evidence from our measurements that such a cost on latency is a result of subnet bandwidth alone. Although we have no way of knowing from our data, we conjecture that the amount of buffering at bottleneck links plays a key role in performance of these applications.

Further details on our experiments and analysis can be found in [Duk10, DCCM10].

## 11. Other Studies

Besides the large scale Internet experiments described above, a number of other studies have been conducted on the effects of IW10 in various environments. These tests were summarized below, with more discussion in Appendix A.

A complete list of tests conducted, with their results and related studies can be found at the [IW10] link.

1. [Sch08] described an earlier evaluation of various Fast Startup approaches, including the "Initial-Start" of 10 MSS.
2. [DCCM10] presented the result from Google's large scale IW10 experiments, with a focus on areas with highly multiplexed links or limited broadband deployment such as Africa and South America.
3. [CW10] contained a testbed study on IW10 performance over slow links. It also studied how short flows with a larger initial window might affect the throughput performance of other co-existing, long lived, bulk data transfers.
4. [Sch11] compared IW10 against a number of other fast startup schemes, and concluded that IW10 works rather well and is also quite fair.
5. [JNDK10] and later [JNDK10-1] studied the effect of IW10 over cellular networks.
6. [AERG11] studied the effect of larger ICW sizes, among other things, on end users' page load time from Yahoo!'s Content Delivery Network.

## 12. Usage and Deployment Recommendations

Further experiments are required before a larger initial window shall be enabled by default in the Internet. The existing measurement results indicate that this does not cause significant harm to other traffic. However, widespread use in the Internet could reveal issues not known yet, e.g., regarding fairness or impact on latency-sensitive traffic such as VoIP.

Therefore, special care is needed when using this experimental TCP extension, in particular on large-scale systems originating a significant amount of Internet traffic, or on large numbers of individual consumer-level systems that have similar aggregate impact. Anyone (stack vendors, network administrators, etc.) turning on a larger initial window SHOULD ensure that the performance is monitored before and after that change. A key metric to monitor is the rate of packet losses, ECN marking, or segment retransmissions during the

initial burst. The sender SHOULD cache such information about connection setups using an initial window larger than allowed by RFC 3390, and new connections SHOULD fall back to the initial window allowed by RFC 3390 if there is evidence of performance issues. Further experiments are needed on the design of such a cache and corresponding heuristics.

Other relevant metrics that may indicate a need to reduce the IW include an increased overall percentage of packet loss or segment retransmissions as well as application-level metrics such as reduced data transfer completion times or impaired media quality.

It is important also to take into account hosts that do not implement a larger initial window. Furthermore, any deployment of IW10 should be aware that there are potential side effects to real-time traffic (such as VoIP). If users observe any significant deterioration of performance, they SHOULD fall back to an initial window as allowed by RFC 3390 for safety reasons. An increased initial window MUST NOT be turned on by default on systems without such monitoring capabilities.

The IETF TCPM working group is very much interested in further reports from experiments with this specification and encourages the publication of such measurement data. By now, there are no adequate studies available that either prove or do not prove impact of IW10 to real-time traffic. Further experimentation in this directions is encouraged.

If no significant harm is reported, a follow-up document may revisit the question on whether a larger initial window can be safely used by default in all Internet hosts. Resolution of these experiments and tighter specifications of the suggestions here might be grounds for a future standards track document on the same topic.

### 13. Related Proposals

Two other proposals [All10, Tou12] have been published to raise TCP's initial window size over a large timescale. Both aim at reducing the uncertain impact of a larger initial window at an Internet wide scale. Moreover, [Tou12] seeks an algorithm to automate the adjustment of IW safely over long haul period.

Although a modest, static increase of IW to 10 may address the near-term need for better web performance, much work is needed from the TCP research community to find a long term solution to the TCP flow startup problem.

### 14. Security Considerations

This document discusses the initial congestion window permitted for TCP connections. Although changing this value may cause more packet loss, it is highly unlikely to lead to a persistent state of network congestion or even a congestion collapse. Hence it does not raise any known new security issues with TCP.

## 15. Conclusion

This document suggests a simple change to TCP that will reduce the application latency over short-lived TCP connections or links with long RTTs (saving several RTTs during the initial slow-start phase) with little or no negative impact over other flows. Extensive tests have been conducted through both testbeds and large data centers with most results showing improved latency with only a small increase in the packet retransmission rate. Based on these results we believe a modest increase of IW to 10 is the best solution for the near-term deployment, while scaling IW over the long run remains a challenge for the TCP research community.

## 16. IANA Considerations

None

## 17. Acknowledgments

Many people at Google have helped to make the set of large scale tests possible. We would especially like to acknowledge Amit Agarwal, Tom Herbert, Arvind Jain and Tiziana Refice for their major contributions.

## Normative References

- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S. and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3390] Allman, M., Floyd, S. and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.
- [RFC5681] Allman, M., Paxson, V. and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC5827] Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J. and P. Hurtig, "Early Retransmit for TCP and SCTP", RFC 5827, May 2010.
- [RFC6298] Paxson, V., Allman, M., Chu, J. and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

## Informative References

- [AKAM10] "The State of the Internet, 3rd Quarter 2009", Akamai Technologies, Inc., January 2010.  
URL=[http://www.akamai.com/html/about/press/releases/2010/press\\_011310\\_1.html](http://www.akamai.com/html/about/press/releases/2010/press_011310_1.html)
- [AERG11] Al-Fares, M., Elmeleegy, K., Reed, B. and I. Gashinsky, "Overclocking the Yahoo! CDN for Faster Web Page Loads", Internet Measurement Conference, November 2011.
- [All00] Allman, M., "A Web Server's View of the Transport Layer", ACM Computer Communication Review, 30(5), October 2000.
- [All10] Allman, M., "Initial Congestion Window Specification", Internet-draft draft-allman-tcpm-bump-initcwnd-00.txt, work in progress, last updated November 2010.
- [Bel10] Belshe, M., "A Client-Side Argument For Changing TCP Slow Start", January, 2010. URL  
[http://sites.google.com/a/chromium.org/dev/spdy/An\\_Argument\\_For\\_Changing\\_TCP\\_Slow\\_Start.pdf](http://sites.google.com/a/chromium.org/dev/spdy/An_Argument_For_Changing_TCP_Slow_Start.pdf)

- [CD10] Chu, J. and N. Dukkipati, "Increasing TCP's Initial Window", Presented to 77th IRTF ICCRG & IETF TCPM working group meetings, March 2010. URL <http://www.ietf.org/proceedings/77/slides/tcpm-4.pdf>
- [Chu09] Chu, J., "Tuning TCP Parameters for the 21st Century", Presented to 75th IETF TCPM working group meeting, July 2009. URL <http://www.ietf.org/proceedings/75/slides/tcpm-1.pdf>.
- [CoDel] Nichols, K. and V. Jacobson, "Controlling Queue Delay", ACM QUEUE, May 6, 2012.
- [CW10] Chu, J. and Wang, Y., "A Testbed Study on IW10 vs IW3", Presented to 79th IETF TCPM working group meeting, Nov. 2010. URL <http://www.ietf.org/proceedings/79/slides/tcpm-0.pdf>.
- [DCCM10] Dukkipati, D., Cheng, Y., Chu, J. and M. Mathis, "Increasing TCP initial window", Presented to 78th IRTF ICCRG working group meeting, July 2010. URL <http://www.ietf.org/proceedings/78/slides/iccr-3.pdf>
- [DGHS07] Dischinger, M., Gummadi, K., Haeberlen, A. and S. Saroiu, "Characterizing Residential Broadband Networks", Internet Measurement Conference, October 24-26, 2007.
- [Duk10] Dukkipati, N., Refice, T., Cheng, Y., Chu, J., Sutin, N., Agarwal, A., Herbert, T. and J. Arvind, "An Argument for Increasing TCP's Initial Congestion Window", ACM SIGCOMM Computer Communications Review, vol. 40 (2010), pp. 27-33. July 2010.
- [FF99] Floyd, S., and K. Fall, "Promoting the Use of End-to-End Congestion Control in the Internet", IEEE/ACM Transactions on Networking, August 1999.
- [FJ93] Floyd, S. and V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, p. 397-413.
- [Get11] Gettys, J., "Bufferbloat: Dark buffers in the Internet", Presented to 80th IETF TSV Area meeting, March 2011. URL <http://www.ietf.org/proceedings/80/slides/tsvarea-1.pdf>
- [Get11-1] Gettys, J., "IW10 Considered Harmful", Internet-draft draft-gettys-iw10-considered-harmful-00, work in progress, August 2011.

- [IOR2009] Labovitz, C., Iekel-Johnson, S., McPherson, D., Oberheide, J. Jahanian, F. and M. Karir, "Atlas Internet Observatory 2009 Annual Report", 47th NANOG Conference, October 2009.
- [IW10] "TCP IW10 links", URL  
<http://code.google.com/speed/protocols/tcpm-IW10.html>
- [Jac88] Jacobson, V., "Congestion Avoidance and Control", Computer Communication Review, vol. 18, no. 4, pp. 314-329, Aug. 1988.
- [JNDK10] Jarvinen, I., Nyrhinen. A., Ding, A. and M. Kojo, "A Simulation Study on Increasing TCP's IW", Presented to 78th IRTF ICCRG working group meeting, July 2010. URL  
<http://www.ietf.org/proceedings/78/slides/iccr-7.pdf>
- [JNDK10-1] Jarvinen, I., Nyrhinen. A., Ding, A. and M. Kojo, "Effect of IW and Initial RTO changes", Presented to 79th IETF TCPM working group meeting, Nov. 2010. URL  
<http://www.ietf.org/proceedings/79/slides/tcpm-1.pdf>
- [LAJW07] Liu, D., Allman, M., Jin, S. and L. Wang, "Congestion Control Without a Startup Phase", Protocols for Fast, Long Distance Networks (PFLDnet) Workshop, February 2007. URL  
<http://www.icir.org/mallman/papers/jumpstart-pfldnet07.pdf>
- [PK98] Padmanabhan V.N. and R. Katz, "TCP Fast Start: A technique for speeding up web transfers", in Proceedings of IEEE Globecom '98 Internet Mini-Conference, 1998.
- [PRAKS02] Partridge, C., Rockwell, D., Allman, M., Krishnan, R. and J. Sterbenz, "A Swifter Start for TCP", Technical Report No. 8339, BBN Technologies, March 2002.
- [RFC2309] Braden, B., Clark, D., Crowcroft, J., Davie, B., Deering, S., Estrin, D., Floyd, S., Jacobson, V., Minshall, G., Partridge, C., Peterson, L., Ramakrishnan, K., Shenker, S., Wroclawski, J. and L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet", RFC 2309, April 1998.
- [RFC2414] Allman, M., Floyd, S. and C. Partridge, "Increasing TCP's Initial Window", RFC 2414, September 1998.
- [RFC3042] Allman, M., Balakrishnan, H. and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.

- [RFC3150] Dawkins, S., Montenegro, G., Kojo, M. and V. Magret, "End-to-end Performance Implications of Slow Links", BCP 0048, July 2001.
- [RFC4782] Floyd, S., Allman, M., Jain, A. and P. Sarolahti, "Quick-Start for TCP and IP", RFC 4782, January 2007.
- [RFC6077] Papadimitriou, D., Welzl, M., Scharf, M. and B. Briscoe, "Open Research Issues in Internet Congestion Control", section 3.4, RFC 6077, February 2011.
- [RJ10] Ramachandran, S. and A. Jain, "Aggregate Statistics of Size Related Metrics of Web Pages metrics", May 2010. URL <http://code.google.com/speed/articles/web-metrics.html>
- [Sch08] Scharf, M., "Quick-Start, Jump-Start, and Other Fast Startup Approaches", Internet Research Task Force ICCRG, November 17, 2008. URL <http://www.ietf.org/proceedings/73/slides/iccr-2.pdf>
- [Sch11] Scharf, M., "Performance and Fairness Evaluation of IW10 and Other Fast Startup Schemes", Internet Research Task Force ICCRG, March 2011. URL <http://www.ietf.org/proceedings/80/slides/iccr-1.pdf>
- [Sch11-1] Scharf, M., "Comparison of end-to-end and network-supported fast startup congestion control schemes", Computer Networks, Feb. 2011. URL <http://dx.doi.org/10.1016/j.comnet.2011.02.002>
- [SPDY] "SPDY: An experimental protocol for a faster web", URL <http://dev.chromium.org/spdy>
- [Ste08] Sounders S., "Roundup on Parallel Connections", High Performance Web Sites blog. March 2008. URL <http://www.stevesouders.com/blog/2008/03/20/roundup-on-parallel-connections>
- [Tou12] Touch, J., "Automating the Initial Window in TCP", Internet-draft draft-touch-tcpm-automatic-iw-03.txt, work in progress, July 16, 2012.
- [VH97] Visweswaraiyah, V. and J. Heidemann, "Improving Restart of Idle TCP Connections", Technical Report 97-661, University of Southern California, November 1997.

## Appendix A - List of Concerns and Corresponding Test Results

Concerns have been raised since this proposal was first published based on a set of large scale experiments. To better understand the impact of a larger initial window in order to confirm or dismiss these concerns, additional tests have been conducted using either large scale clusters, simulations, or real testbeds. The following attempts to compile the list of concerns and summarize findings from relevant tests.

- o How complete are various tests in covering many different traffic patterns?

The large scale Internet experiments conducted at Google front-end infrastructure covered a large portfolio of services beyond web search. It includes Gmail, Google Maps, Photos, News, Sites, Images, ..., etc, covering a wide variety of traffic sizes and patterns. One notable exception is YouTube because we don't think the large initial window will have much material impact, either positive or negative, on bulk data services.

[CW10] contains some result from a testbed study on how short flows with a larger initial window might affect the throughput performance of other co-existing, long lived, bulk data transfers.

- o Larger bursts from the increase in the initial window cause significantly more packet drops

All the tests conducted on this subject [Duk10, Sch11, Sch11-1, CW10] so far have shown only modest increase on packet drops. The only exception is from the testbed study [CW10] when under extremely high load and/or simultaneous opens. But under those conditions both IW=3 and IW=10 suffered very high packet loss rates though.

- o A large initial window may severely impact TCP performance over highly multiplexed links still common in developing regions

Our large scale experiments described in section 10 above also covered Africa and South America. Measurement data from those regions [DCCM10] revealed improved latency even for those services that employ multiple simultaneous connections, at the cost of small increase in the retransmission rate. It seems that the round trip savings from a larger initial window more than make up the time spent on recovering more lost packets.

Similar phenomenon have also been observed from testbed study [CW10].

- o Why 10 segments?

Questions have been raised on how the number 10 was picked. We have tried different sizes in our large scale experiments, and found that 10 segments seem to give most of the benefits for the services we tested while not causing significant increase in the retransmission rates. Going forward 10 segments may turn out to be too small when the average of web object sizes continue to grow. But a scheme to right size the initial window automatically over long timescales has yet to be developed.

- o Need more thorough analysis of the impact on slow links

Although [Duk10] showed the large initial window reduced the average latency even for the dialup link class of only 56Kbps in bandwidth, more studies were needed in order to understand the effect of IW10 on slow links at the microscopic level. [CW10] was conducted for this purpose.

Testbeds in [CW10] emulated a 300ms RTT, bottleneck link bandwidth as low as 64Kbps, and route queue size as low as 40 packets. A large combination of test parameters were used. Almost all tests showed varying degree of latency improvement from IW=10, with only a modest increase in the packet drop rate until a very high load was injected. The testbed result was consistent with both the large scale data center experiments [CD10, DCCM10] and a separate study using NSC simulations [Sch11, Sch11-1].

- o How will the larger initial window affect flows with initial windows 4KB or less?

Flows with the larger initial window will likely grab more bandwidth from a bottleneck link when competing against flows with smaller initial window, at least initially. How long will this "unfairness" last? Will there be any "capture effect" where flows with larger initial window possess a disproportional share of bandwidth beyond just a few round trips?

If there is any "unfairness" issue from flows with different initial windows, it did not show up in the large scale experiments, as the average latency for the bucket of all responses < 4KB did not seem to be affected by the presence of many other larger responses employing large initial window. As a matter of fact they seemed to benefit from the large initial window too, as shown in Figure 7 of [Duk10].

The same phenomenon seems to exist in the testbed experiments [CW10]. Flows with IW=3 only suffered slightly when competing

against flows with IW=10 in light to median loads. Under high load both flows' latency improved when mixed together. Also long-lived, background bulk-data flows seemed to enjoy higher throughput when running against many foreground short flows of IW=10 than against short flows of IW=3. One plausible explanation was IW=10 enabled short flows to complete sooner, leaving more room for the long-lived, background flows.

A study using NSC simulator has also concluded that IW=10 works rather well and is quite fair against IW=3 [Sch11, Sch11-1].

- o How will a larger initial window perform over cellular networks?

Some simulation studies [JNDK10, JNDK10-1] have been conducted to study the effect of a larger initial window on wireless links from 2G to 4G networks (EGDE/HSPA/LTE). The overall result seems mixed in both raw performance and the fairness index.

Author's Addresses

Jerry Chu  
Google, Inc.  
1600 Amphitheatre Parkway  
Mountain View, CA 94043  
USA  
EMail: [hkchu@google.com](mailto:hkchu@google.com)

Nandita Dukkipati  
Google, Inc.  
1600 Amphitheatre Parkway  
Mountain View, CA 94043  
USA  
EMail: [nanditad@google.com](mailto:nanditad@google.com)

Yuchung Cheng  
Google, Inc.  
1600 Amphitheatre Parkway  
Mountain View, CA 94043  
USA  
EMail: [ycheng@google.com](mailto:ycheng@google.com)

Matt Mathis  
Google, Inc.  
1600 Amphitheatre Parkway  
Mountain View, CA 94043  
USA  
EMail: [mattmathis@google.com](mailto:mattmathis@google.com)

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.

TCP Maintenance Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: August 10, 2013

M. Mathis  
N. Dukkipati  
Y. Cheng  
Google, Inc  
Feb 6, 2013

Proportional Rate Reduction for TCP  
draft-ietf-tcpm-proportional-rate-reduction-04.txt

Abstract

This document describes an experimental algorithm, Proportional Rate Reduction (PPR) to improve the accuracy of the amount of data sent by TCP during loss recovery. Standard Congestion Control requires that TCP and other protocols reduce their congestion window in response to losses. This window reduction naturally occurs in the same round trip as the data retransmissions to repair the losses, and is implemented by choosing not to transmit any data in response to some ACKs arriving from the receiver. Two widely deployed algorithms are used to implement this window reduction: Fast Recovery and Rate Halving. Both algorithms are needlessly fragile under a number of conditions, particularly when there is a burst of losses such that the number of ACKs returning to the sender is small. Proportional Rate Reduction minimizes these excess window adjustments such that at the end of recovery the actual window size will be as close as possible to ssthresh, the window size determined by the congestion control algorithm. It is patterned after Rate Halving, but using the fraction that is appropriate for target window chosen by the congestion control algorithm.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 10, 2013.

## Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Definitions . . . . .	5
3. Algorithms . . . . .	6
3.1. Examples . . . . .	6
4. Properties . . . . .	9
5. Measurements . . . . .	11
6. Conclusion and Recommendations . . . . .	12
7. Acknowledgements . . . . .	13
8. Security Considerations . . . . .	13
9. IANA Considerations . . . . .	14
10. References . . . . .	14
10.1. Normative References . . . . .	14
10.2. Informative References . . . . .	14
Appendix A. Strong Packet Conservation Bound . . . . .	15
Authors' Addresses . . . . .	16

## 1. Introduction

This document describes an experimental algorithm, Proportional Rate Reduction (PPR) to improve the accuracy of the amount of data sent by TCP during loss recovery.

Standard Congestion Control [RFC5681] requires that TCP (and other protocols) reduce their congestion window in response to losses. Fast Recovery, described in the same document, is the reference algorithm for making this adjustment. Its stated goal is to recover TCP's self clock by relying on returning ACKs during recovery to clock more data into the network. Fast Recovery typically adjusts the window by waiting for one half RTT of ACKs to pass before sending any data. It is fragile because it can not compensate for the implicit window reduction caused by the losses themselves.

RFC 6675 [RFC6675] makes Fast Recovery with SACK [RFC2018] more accurate by computing "pipe", a sender side estimate of the number of bytes still outstanding in the network. With RFC 6675, Fast Recovery is implemented by sending data as necessary on each ACK to prevent pipe from falling below ssthresh, the window size as determined by the congestion control algorithm. This protects Fast Recovery from timeouts in many cases where there are heavy losses, although not if the entire second half of the window of data or ACKs are lost. However, a single ACK carrying a SACK option that implies a large quantity of missing data can cause a step discontinuity in the pipe estimator, which can cause Fast Retransmit to send a burst of data.

The rate-halving algorithm sends data on alternate ACKs during recovery, such that after one RTT the window has been halved. Rate-halving is implemented in Linux after only being informally published [RHweb], including an uncompleted Internet-Draft [RHID]. Rate-halving also does not adequately compensate for the implicit window reduction caused by the losses and assumes a net 50% window reduction, which was completely standard at the time it was written, but not appropriate for modern congestion control algorithms such as Cubic [CUBIC], which reduce the window by less than 50%. As a consequence rate-halving often allows the window to fall further than necessary, reducing performance and increasing the risk of timeouts if there are additional losses.

Proportional Rate Reduction (PPR) avoids these excess window adjustments such that at the end of recovery the actual window size will be as close as possible to ssthresh, the window size determined by the congestion control algorithm. It is patterned after Rate Halving, but using the fraction that is appropriate for the target window chosen by the congestion control algorithm. During PPR one of two additional reduction bound algorithms limits the total window

reduction due to all mechanisms, including transient application stalls and the losses themselves.

We describe two slightly different reduction bound algorithms: conservative reduction bound (CRB), which is strictly packet conserving; and a slow start reduction bound (SSRB), which is more aggressive than CRB by at most one segment per ACK. PRR-CRB meets the Strong Packet Conservation Bound described in Appendix A, however in real networks it does not perform as well as the algorithms described in RFC 6675, which prove to be more aggressive in a significant number of cases. SSRB offers a compromise by allowing TCP to send one additional segment per ACK relative to CRB in some situations. Although SSRB is less aggressive than RFC 6675 (transmitting fewer segments or taking more time to transmit them) it outperforms it, due to the lower probability of additional losses during recovery.

The Strong Packet Conservation Bound on which PRR and both reduction bounds are based is patterned after Van Jacobson's packet conservation principle: segments delivered to the receiver are used as the clock to trigger sending the same number of segments back into the network. As much as possible Proportional Rate Reduction and the reduction bound algorithms rely on this self clock process, and are only slightly affected by the accuracy of other estimators, such as pipe [RFC6675] and cwnd. This is what gives the algorithms their precision in the presence of events that cause uncertainty in other estimators.

The original definition of the packet conservation principle [Jacobson88] treated packets that are presumed to be lost (e.g. marked as candidates for retransmission) as having left the network. This idea is reflected in the pipe estimator defined in RFC 6675 and used here, but it is distinct from Strong Packet Conservation Bound described in Appendix A, which is defined solely on the basis of data arriving at the receiver.

We evaluated these and other algorithms in a large scale measurement study presented in a companion paper [IMC11] and summarized in Section 5. This measurement study was based on RFC 3517 [RFC3517], which has since been superseded by RFC 6675. Since there are slight difference between the two specifications, and we were meticulous about our implementation of RFC 3517 we are not comfortable unconditionally asserting that our measurement results apply to RFC 6675, although we believe this to be the case. We have instead chosen to be pedantic about describing measurement results relative to RFC 3517, on which they were actually based. General discussions algorithms and their properties have been updated to refer to RFC 6675.

We found that for authentic network traffic PRR+SSRB outperforms both RFC 3517 and Linux Rate Halving even though it is less aggressive than RFC 3517. We believe that these results apply to RFC 6675 as well.

The algorithms are described as modifications to RFC 5681 [RFC5681], TCP Congestion Control, using concepts drawn from the pipe algorithm [RFC6675]. They are most accurate and more easily implemented with SACK [RFC2018], but do not require SACK.

## 2. Definitions

The following terms, parameters and state variables are used as they are defined in earlier documents:

RFC 793: `snd.una`

RFC 5681: duplicate ACK, FlightSize, Sender Maximum Segment Size (SMSS)

RFC 6675: covered (as in "covered sequence numbers")

Voluntary window reductions: choosing not to send data in response to some ACKs, for the purpose of reducing the sending window size and data rate.

We define some additional variables:

SACKd: The total number of bytes that the scoreboard indicates have been delivered to the receiver. This can be computed by scanning the scoreboard and counting the total number of bytes covered by all sack blocks. If SACK is not in use, SACKd is not defined.

DeliveredData: The total number of bytes that the current ACK indicates have been delivered to the receiver. When not in recovery, DeliveredData is the change in `snd.una`. With SACK, DeliveredData can be computed precisely as the change in `snd.una` plus the (signed) change in SACKd. In recovery without SACK, DeliveredData is estimated to be 1 SMSS on duplicate acknowledgements, and on a subsequent partial or full ACK, DeliveredData is estimated to be the change in `snd.una`, minus one SMSS for each preceding duplicate ACK.

Note that DeliveredData is robust: for TCP using SACK, DeliveredData can be precisely computed anywhere in the network just by inspecting the returning ACKs. The consequence of missing ACKs is that later ACKs will show a larger DeliveredData. Furthermore, for any TCP (with or without SACK) the sum of DeliveredData must agree with the

forward progress over the same time interval.

We introduce a local variable "sndcnt", which indicates exactly how many bytes should be sent in response to each ACK. Note that the decision of which data to send (e.g. retransmit missing data or send more new data) is out of scope for this document.

### 3. Algorithms

At the beginning of recovery initialize PRR state. This assumes a modern congestion control algorithm, CongCtrlAlg(), that might set ssthresh to something other than FlightSize/2:

```
ssthresh = CongCtrlAlg() // Target cwnd after recovery
prrr_delivered = 0       // Total bytes delivered during recovery
prrr_out = 0             // Total bytes sent during recovery
RecoverFS = snd.nxt-snd.una // FlightSize at the start of recovery
```

On every ACK during recovery compute:

```
DeliveredData = change_in(snd.una) + change_in(SACKd)
prrr_delivered += DeliveredData
pipe = (RFC 6675 pipe algorithm)
if (pipe > ssthresh) {
    // Proportional Rate Reduction
    sndcnt = CEIL(prrr_delivered * ssthresh / RecoverFS) - prrr_out
} else {
    // Two version of the reduction bound
    if (conservative) { // PRR+CRB
        limit = prrr_delivered - prrr_out
    } else { // PRR+SSRB
        limit = MAX(prrr_delivered - prrr_out, DeliveredData) + MSS
    }
    // Attempt to catch up, as permitted by limit
    sndcnt = MIN(ssthresh - pipe, limit)
}
```

On any data transmission or retransmission:

```
prrr_out += (data sent) // strictly less than or equal to sndcnt
```

#### 3.1. Examples

We illustrate these algorithms by showing their different behaviors for two scenarios: TCP experiencing either a single loss or a burst of 15 consecutive losses. In all cases we assume bulk data (no application pauses), standard AIMD congestion control and cwnd =

FlightSize = pipe = 20 segments, so ssthresh will be set to 10 at the beginning of recovery. We also assume standard Fast Retransmit and Limited Transmit [RFC3042], so TCP will send two new segments followed by one retransmit in response to the first 3 duplicate ACKs following the losses.

Each of the diagrams below shows the per ACK response to the first round trip for the various recovery algorithms when the zeroth segment is lost. The top line indicates the transmitted segment number triggering the ACKs, with an X for the lost segment. "cwnd" and "pipe" indicate the values of these algorithms after processing each returning ACK. "Sent" indicates how much 'N'ew or 'R'etransmitted data would be sent. Note that the algorithms for deciding which data to send are out of scope of this document.

When there is a single loss, PRR with either of the reduction bound algorithms has the same behavior. We show "RB", a flag indicating which reduction bound subexpression ultimately determined the value of sndcnt. When there is minimal losses "limit" (both algorithms) will always be larger than ssthresh - pipe, so the sndcnt will be ssthresh - pipe indicated by "s" in the "RB" row.

#### RFC 6675

ack#	X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
cwnd:		20	20	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
pipe:		19	19	18	18	17	16	15	14	13	12	11	10	10	10	10	10	10	10	10
sent:		N	N	R										N	N	N	N	N	N	N

#### Rate Halving (Linux)

ack#	X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
cwnd:		20	20	19	18	18	17	17	16	16	15	15	14	14	13	13	12	12	11	11
pipe:		19	19	18	18	17	17	16	16	15	15	14	14	13	13	12	12	11	11	10
sent:		N	N	R		N		N		N		N		N		N		N		N

#### PRR

ack#	X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
pipe:		19	19	18	18	18	17	17	16	16	15	15	14	14	13	13	12	12	11	10
sent:		N	N	R		N		N		N		N		N		N			N	N
RB:																			s	s

Cwnd is not shown because PRR does not use it.

#### Key for RB

s:	sndcnt = ssthresh - pipe	// from ssthresh
b:	sndcnt = prr_delivered - prr_out + SMSS	// from banked
d:	sndcnt = DeliveredData + SMSS	// from DeliveredData

(Sometimes more than one applies)

Note that all three algorithms send the same total amount of data. RFC 6675 experiences a "half-window of silence", while the Rate Halving and PRR spread the voluntary window reduction across an entire RTT.

Next we consider the same initial conditions when the first 15 packets (0-14) are lost. During the remainder of the lossy RTT, only 5 ACKs are returned to the sender. We examine each of these algorithms in succession.

#### RFC 6675

ack#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
cwnd:																20	20	11	11	11
pipe:																19	19	4	10	10
sent:																N	N	7R	R	R

#### Rate Halving (Linux)

ack#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
cwnd:																20	20	5	5	5
pipe:																19	19	4	4	4
sent:																N	N	R	R	R

#### PRR-CRB

ack#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
pipe:																19	19	4	4	4
sent:																N	N	R	R	R
RB:																		b	b	b

#### PRR-SSRB

ack#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
pipe:																19	19	4	5	6
sent:																N	N	2R	2R	2R
RB:																		bd	d	d

In this specific situation, RFC 6675 is more aggressive, because once fast retransmit is triggered (on the ACK for segment 17) TCP immediately retransmits sufficient data to bring pipe up to cwnd. Our measurement data (see Section 5) indicates that RFC 6675 significantly outperforms Rate Halving, PRR-CRB and some other similarly conservative algorithms that we tested, showing that it is significantly common for the actual losses to exceed the window

reduction determined by the congestion control algorithm.

The Linux implementation of Rate Halving includes an early version of the conservative reduction bound [RHweb]. In this situation the five ACKs trigger exactly one transmission each (2 new data, 3 old data), and cwnd is set to 5. At a window size of 5, it takes three round trips to retransmit all 15 lost segments. Rate Halving does not raise the window at all during recovery, so when recovery finally completes, TCP will slowstart cwnd from 5 up to 10. In this example, TCP operates at half of the window chosen by the congestion control for more than three RTTs, increasing the elapsed time and exposing it to timeouts in the event that there are additional losses.

PRR-CRB implements a conservative reduction bound. Since the total losses bring pipe below ssthresh, data is sent such that the total data transmitted, prr\_out, follows the total data delivered to the receiver as reported by returning ACKs. Transmission is controlled by the sending limit, which was set to prr\_delivered - prr\_out. This is indicated by the RB:b tagging in the figure. In this case PRR-CRB is exposed to exactly the same problems as Rate Halving, the excess window reduction causes it to take excessively long to recover the losses and exposes it to additional timeouts.

PRR-SSRB increases the window by exactly 1 segment per ACK until pipe rises to ssthresh during recovery. This is accomplished by setting limit to one greater than the data reported to have been delivered to the receiver on this ACK, implementing slowstart during recovery, and indicated by RB:d tagging in the figure. Although increasing the window during recovery seems to be ill advised, it is important to remember that this is actually less aggressive than permitted by RFC 5681, which sends the same quantity of additional data as a single burst in response to the ACK that triggered Fast Retransmit

For less extreme events, where the total losses are smaller than the difference between Flight Size and ssthresh, PRR-CRB and PRR-SSRB have identical behaviours.

#### 4. Properties

The following properties are common to both PRR-CRB and PRR-SSRB except as noted:

Proportional Rate Reduction maintains TCPs ACK clocking across most recovery events, including burst losses. RFC 6675 can send large unclocked bursts following burst losses.

Normally Proportional Rate Reduction will spread voluntary window

reductions out evenly across a full RTT. This has the potential to generally reduce the burstiness of Internet traffic, and could be considered to be a type of soft pacing. Hypothetically, any pacing increases the probability that different flows are interleaved, reducing the opportunity for ACK compression and other phenomena that increase traffic burstiness. However these effects have not been quantified.

If there are minimal losses, Proportional Rate Reduction will converge to exactly the target window chosen by the congestion control algorithm. Note that as TCP approaches the end of recovery `pr_r_delivered` will approach `RecoverFS` and `sndcnt` will be computed such that `pr_r_out` approaches `ssthresh`.

Implicit window reductions due to multiple isolated losses during recovery cause later voluntary reductions to be skipped. For small numbers of losses the window size ends at exactly the window chosen by the congestion control algorithm.

For burst losses, earlier voluntary window reductions can be undone by sending extra segments in response to ACKs arriving later during recovery. Note that as long as some voluntary window reductions are not undone, the final value for `pipe` will be the same as `ssthresh`, the target `cwnd` value chosen by the congestion control algorithm.

Proportional Rate Reduction with either reduction bound improves the situation when there are application stalls (e.g. when the sending application does not queue data for transmission quickly enough or the receiver stops advancing `rwnd`). When there is an application stall early during recovery `pr_r_out` will fall behind the sum of the transmissions permitted by `sndcnt`. The missed opportunities to send due to stalls are treated like banked voluntary window reductions: specifically they cause `pr_r_delivered-pr_r_out` to be significantly positive. If the application catches up while TCP is still in recovery, TCP will send a partial window burst to catch up to exactly where it would have been, had the application never stalled. Although this burst might be viewed as being hard on the network, this is exactly what happens every time there is a partial RTT application stall while not in recovery. We have made the partial RTT stall behavior uniform in all states. Changing this behavior is out of scope for this document.

Proportional Rate Reduction with Reduction Bound is less sensitive to errors in the pipe estimator. While in recovery, `pipe` is intrinsically an estimator, using incomplete information to estimate if un-SACKed segments are actually lost or merely out-of-order in the network. Under some conditions `pipe` can have significant errors, for example `pipe` is underestimated when a burst of reordered data is

prematurely assumed to be lost and marked for retransmission. If the transmissions are regulated directly by pipe as they are with RFC 6675, such as step discontinuity in the pipe estimator causes a burst of data, which can not be retracted once the pipe estimator is corrected a few ACKs later. For PRR, pipe merely determines which algorithm, Proportional Rate Reduction or the reduction bound, is used to compute `sndcnt` from `DeliveredData`. While pipe is underestimated the algorithms are different by at most one segment per ACK. Once pipe is updated they converge to the same final window at the end of recovery.

Under all conditions and sequences of events during recovery, PRR-CRB strictly bounds the data transmitted to be equal to or less than the amount of data delivered to the receiver. We claim that this Strong Packet Conservation Bound is the most aggressive algorithm that does not lead to additional forced losses in some environments. It has the property that if there is a standing queue at a bottleneck with no cross traffic, the queue will maintain exactly constant length for the duration of the recovery, except for  $\pm 1$  fluctuation due to differences in packet arrival and exit times. See Appendix A for a detailed discussion of this property.

Although the Strong Packet Conserving Bound is very appealing for a number of reasons, our measurements summarized in Section 5 demonstrate that it is less aggressive and does not perform as well as RFC 6675, which permits large bursts of data when there are bursts of losses. PRR-SSRB is a compromise that permits TCP to send one extra segment per ACK as compared to the packet conserving bound. From the perspective of a strict packet conserving bound, PRR-SSRB does indeed open the window during recovery, however it is significantly less aggressive than RFC6675 in the presence of burst losses.

## 5. Measurements

In a companion IMC11 paper [IMC11] we describe some measurements comparing the various strategies for reducing the window during recovery. The experiments were performed on servers carrying Google production traffic and are briefly summarized here.

The various window reduction algorithms and extensive instrumentation were all implemented in Linux 2.6. We used the uniform set of algorithms present in the base Linux implementation, including CUBIC [CUBIC], limited transmit [RFC3042], threshold transmit from [FACK] (this algorithm was not present in RFC 3517, but a similar algorithm has been added to RFC 6675) and lost retransmission detection algorithms. We confirmed that the behaviors of Rate Halving (the

Linux default), RFC 3517 and PRR were authentic to their respective specifications and that performance and features were comparable to the kernels in production use. All of the different window reduction algorithms were all present in a common kernel and could be selected with a `sysctl`, such that we had an absolutely uniform baseline for comparing them.

Our experiments included an additional algorithm, PRR with an unlimited bound (PRR-UB), which sends `ssthresh-pipe` bursts when pipe falls below `ssthresh`. This behavior parallels RFC 3517.

An important detail of this configuration is that CUBIC only reduces the window by 30%, as opposed to the 50% reduction used by traditional congestion control algorithms. This accentuates the tendency for RFC 3517 and PRR-UB to send a burst at the point when Fast Retransmit gets triggered because pipe is likely to already be below `ssthresh`. Precisely this condition was observed for 32% of the recovery events: pipe fell below `ssthresh` before Fast Retransmit is triggered, thus the various PRR algorithms start in the reduction bound phase, and RFC 3517 sends bursts of segments with the fast retransmit.

In the companion paper we observe that PRR-SSRB spends the least time in recovery of all the algorithms tested, largely because it experiences fewer timeouts once it is already in recovery.

RFC 3517 experiences 29% more detected lost retransmissions and 2.6% more timeouts (presumably due to undetected lost retransmissions) than PRR-SSRB. These results are representative of PRR-UB and other algorithms that send bursts when pipe falls below `ssthresh`.

Rate Halving experiences 5% more timeouts and significantly smaller final `cwnd` values at the end of recovery. The smaller `cwnd` sometimes causes the recovery itself to take extra round trips. These results are representative of PRR-CRB and other algorithms that implement strict packet conservation during recovery.

## 6. Conclusion and Recommendations

Although the Strong Packet Conserving Bound used in PRR-CRB is very appealing for a number of reasons, our measurements show that it is less aggressive and does not perform as well as RFC 3517, (and by implication RFC 6675), which permit bursts of data when there are bursts of losses. RFC 3517 and RFC 6675 are conservative in the original sense of Van Jacobson's packet conservation principle, which included the assumption that presumed lost segments have indeed left the network. PRR-CRB makes no such assumption, following instead a

Strong Packet Conserving Bound, in which only packets that have actually arrived at the receiver are considered to have left the network. PRR-SSRB is a compromise that permits TCP to send one extra segment per ACK relative to the Strong Packet Conserving Bound, to partially compensate for excess losses.

From the perspective of the Strong Packet Conserving Bound, PRR-SSRB does indeed open the window during recovery, however it is significantly less aggressive than RFC 3517 (and RFC 6675) in the presence of burst losses. Even so, it often outperforms RFC 3517, (and presumably RFC 6675) because it avoids some of the self inflicted losses caused by bursts.

At this time we see no reason not to test and deploy PRR-SSRB on a large scale. Implementers worried about any potential impact of raising the window during recovery may want to optionally support PRR-CRB (which is actually simpler to implement) for comparison studies. Furthermore, there is one minor detail of PRR that can be improved by replacing `pipe` by `total_pipe` as defined by Laminar TCP [Laminar].

One final comment about terminology: we expect that common usage will drop "slow start reduction bound" from the algorithm name. This document needed to be pedantic about having distinct names for proportional rate reduction and every variant of the reduction bound. However, we do not anticipate any future exploration of the alternative reduction bounds.

## 7. Acknowledgements

This draft is based in part on previous incomplete work by Matt Mathis, Jeff Semke and Jamshid Mahdavi [RHID] and influenced by several discussion with John Heffner.

Monia Ghobadi and Sivasankar Radhakrishnan helped analyze the experiments.

Ilpo Jarvinen reviewed the code.

Mark Allman improved the document through his insightful review.

## 8. Security Considerations

Proportional Rate Reduction does not change the risk profile for TCP.

Implementers that change PRR from counting bytes to segments have to

be cautious about the effects of ACK splitting attacks [Savage99], where the receiver acknowledges partial segments for the purpose of confusing the sender's congestion accounting.

## 9. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

## 10. References

### 10.1. Normative References

- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 2012.

### 10.2. Informative References

- [RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.
- [RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC 3517, April 2003.
- [IMC11] Dukkkipati, N., Mathis, M., and Y. Cheng, "Proportional Rate Reduction for TCP", ACM Internet Measurement Conference IMC11, December 2011.
- [FACK] Mathis, M. and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", ACM SIGCOMM SIGCOMM96, August 1996.
- [RHID] Mathis, M., Semke, J., Mahdavi, J., and K. Lahey, "The Rate-Halving Algorithm for TCP Congestion Control",

draft-mathis-tcp-ratehalving (work in progress),  
June 1999.

[RHweb] Mathis, M. and J. Mahdavi, "TCP Rate-Halving with Bounding Parameters", Web publication , December 1997.

[CUBIC] Rhee, I. and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant", PFLDnet 2005, Feb 2005.

[Jacobson88]  
Jacobson, V., "Congestion Avoidance and Control", SIGCOMM Comput. Commun. Rev. 18(4), Aug 1988.

[Savage99]  
Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP congestion control with a misbehaving receiver", SIGCOMM Comput. Commun. Rev. 29(5), October 1999.

[Laminar] Mathis, M., "Laminar TCP and the case for refactoring TCP congestion control", draft-mathis-tcpm-tcp-laminar-01 (work in progress), July 2012.

#### Appendix A. Strong Packet Conservation Bound

PRR-CRB is based on a conservative, philosophically pure and aesthetically appealing Strong Packet Conservation Bound, described here. Although inspired by Van Jacobson's packet conservation principle [Jacobson88], it differs in how it treats segments that are missing and presumed lost. Under all conditions and sequences of events during recovery, PRR-CRB strictly bounds the data transmitted to be equal to or less than the amount of data delivered to the receiver. Note that the effects of presumed losses are included in the pipe calculation, but do not affect the outcome of PRR-CRB, once pipe has fallen below ssthresh.

We claim that this Strong Packet Conservation Bound is the most aggressive algorithm that does not lead to additional forced losses in some environments. It has the property that if there is a standing queue at a bottleneck that is carrying no other traffic, the queue will maintain exactly constant length for the entire duration of the recovery, except for  $\pm 1$  fluctuation due to differences in packet arrival and exit times. Any less aggressive algorithm will result in a declining queue at the bottleneck. Any more aggressive algorithm will result in an increasing queue or additional losses if it is a full drop tail queue.

We demonstrate this property with a little thought experiment:

Imagine a network path that has insignificant delays in both directions, except for the processing time and queue at a single bottleneck in the forward path. By insignificant delay, we mean when a packet is "served" at the head of the bottleneck queue, the following events happen in much less than one bottleneck packet time: the packet arrives at the receiver; the receiver sends an ACK; which arrives at the sender; the sender processes the ACK and sends some data; the data is queued at the bottleneck.

If `sndcnt` is set to `DeliveredData` and nothing else is inhibiting sending data, then clearly the data arriving at the bottleneck queue will exactly replace the data that was served at the head of the queue, so the queue will have a constant length. If queue is drop tail and full then the queue will stay exactly full. Losses or reordering on the ACK path only cause wider fluctuations in the queue size, but do not raise its peak size, independent of whether the data is in order or out-of-order (including loss recovery from an earlier RTT). Any more aggressive algorithm which sends additional data will overflow the drop tail queue and cause loss. Any less aggressive algorithm will under fill the queue. Therefore setting `sndcnt` to `DeliveredData` is the most aggressive algorithm that does not cause forced losses in this simple network. Relaxing the assumptions (e.g. making delays more authentic and adding more flows, delayed ACKs, etc) are likely to increase the fine grained fluctuations in queue size but do not change its basic behavior.

Note that the congestion control algorithm implements a broader notion of optimal that includes appropriately sharing the network. Typical congestion control algorithms are likely to reduce the data sent relative to the packet conserving bound implemented by PRR bringing TCP's actual window down to `ssthresh`.

#### Authors' Addresses

Matt Mathis  
Google, Inc  
1600 Amphitheater Parkway  
Mountain View, California 93117  
USA

Email: [mattmathis@google.com](mailto:mattmathis@google.com)

Nandita Dukkipati  
Google, Inc  
1600 Amphitheater Parkway  
Mountain View, California 93117  
USA

Email: nanditad@google.com

Yuchung Cheng  
Google, Inc  
1600 Amphitheater Parkway  
Mountain View, California 93117  
USA

Email: ycheng@google.com



TCP Maintenance & Minor Extensions (tcpm)  
Internet-Draft  
Intended status: Experimental  
Expires: April 21, 2016

B. Briscoe  
Simula Research Laboratory  
M. Kuehlewind  
ETH Zurich  
R. Scheffenegger  
NetApp, Inc.  
October 19, 2015

More Accurate ECN Feedback in TCP  
draft-kuehlewind-tcpm-accurate-ecn-05

Abstract

Explicit Congestion Notification (ECN) is a mechanism where network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the end-points. Receivers with an ECN-capable transport protocol feed back this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recently, new TCP mechanisms like Congestion Exposure (ConEx) or Data Center TCP (DCTCP) need more accurate ECN feedback information whenever more than one marking is received in one RTT. This document specifies an experimental scheme to provide more than one feedback signal per RTT in the TCP header. Given TCP header space is scarce, it overloads the three existing ECN-related flags in the TCP header and provides additional information in a new TCP option.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 21, 2016.

## Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Document Roadmap . . . . .	4
1.2. Goals . . . . .	4
1.3. Experiment Goals . . . . .	5
1.4. Terminology . . . . .	5
1.5. Recap of Existing ECN feedback in IP/TCP . . . . .	6
2. AcceCN Protocol Overview and Rationale . . . . .	7
2.1. Capability Negotiation . . . . .	8
2.2. Feedback Mechanism . . . . .	8
2.3. Delayed ACKs and Resilience Against ACK Loss . . . . .	9
2.4. Feedback Metrics . . . . .	10
2.5. Generic (Dumb) Reflector . . . . .	10
3. AcceCN Protocol Specification . . . . .	11
3.1. Negotiation during the TCP handshake . . . . .	11
3.2. AcceCN Feedback . . . . .	14
3.2.1. The ACE Field . . . . .	14
3.2.2. Safety against Ambiguity of the ACE Field . . . . .	16
3.2.3. The AcceCN Option . . . . .	16
3.2.4. Path Traversal of the AcceCN Option . . . . .	17
3.2.5. Usage of the AcceCN TCP Option . . . . .	19
3.3. AcceCN Compliance by TCP Proxies, Offload Engines and other Middleboxes . . . . .	20
4. Interaction with Other TCP Variants . . . . .	21
4.1. Compatibility with SYN Cookies . . . . .	21
4.2. Compatibility with Other TCP Options and Experiments . . . . .	21
4.3. Compatibility with Feedback Integrity Mechanisms . . . . .	21
5. Protocol Properties . . . . .	23
6. IANA Considerations . . . . .	25
7. Security Considerations . . . . .	25
8. Acknowledgements . . . . .	26
9. Comments Solicited . . . . .	26

10. References	26
10.1. Normative References	26
10.2. Informative References	27
Appendix A. Example Algorithms	29
A.1. Example Algorithm to Encode/Decode the AcceCN Option	29
A.2. Example Algorithm for Safety Against Long Sequences of ACK Loss	30
A.2.1. Safety Algorithm without the AcceCN Option	30
A.2.2. Safety Algorithm with the AcceCN Option	32
A.3. Example Algorithm to Estimate Marked Bytes from Marked Packets	33
A.4. Example Algorithm to Beacon AcceCN Options	34
A.5. Example Algorithm to Count Not-ECT Bytes	35
Appendix B. Alternative Design Choices (To Be Removed Before Publication)	35
Appendix C. Open Protocol Design Issues (To Be Removed Before Publication)	36
Appendix D. Changes in This Version (To Be Removed Before Publication)	37
Authors' Addresses	37

## 1. Introduction

Explicit Congestion Notification (ECN) [RFC3168] is a mechanism where network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the end-points. Receivers with an ECN-capable transport protocol feed back this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recently, proposed mechanisms like Congestion Exposure (ConEx [I-D.ietf-conex-abstract-mech]) or DCTCP [I-D.bensley-tcpm-dctcp] need more accurate ECN feedback information whenever more than one marking is received in one RTT. A fuller treatment of the motivation for this specification is given in the associated requirements document [RFC7560].

This document specifies an experimental scheme for ECN feedback in the TCP header to provide more than one feedback signal per RTT. It will be called the more accurate ECN feedback scheme, or AcceCN for short. If AcceCN progresses from experimental to the standards track, it is intended to be a complete replacement for classic ECN feedback, not a fork in the design of TCP. Thus, the applicability of AcceCN is intended to include all public and private IP networks (and even any non-IP networks over which TCP is used today). Until the AcceCN experiment succeeds, [RFC3168] will remain as the standards track specification for adding ECN to TCP. To avoid confusion, in this document we use the term 'classic ECN' for the pre-existing ECN specification [RFC3168].

AcceECN is solely an (experimental) change to the TCP wire protocol. It is completely independent of how TCP might respond to congestion feedback. This specification overloads flags and fields in the main TCP header with new definitions, so both ends have to support the new wire protocol before it can be used. Therefore during the TCP handshake the two ends use the three ECN-related flags in the TCP header to negotiate the most advanced feedback protocol that they can both support.

It is likely (but not required) that the AcceECN protocol will be implemented along with the following experimental additions to the TCP-ECN protocol: ECN-capable SYN/ACK [RFC5562], ECN path-probing and fall-back [I-D.kuehlewind-tcpm-ecn-fallback] and testing receiver non-compliance [I-D.moncaster-tcpm-rcv-cheat].

### 1.1. Document Roadmap

The following introductory sections outline the goals of AcceECN (Section 1.2) and the goal of experiments with ECN (Section 1.3) so that it is clear what success would look like. Then terminology is defined (Section 1.4) and a recap of existing prerequisite technology is given (Section 1.5).

Section 2 gives an informative overview of the AcceECN protocol. Then Section 3 gives the normative protocol specification. Section 4 assesses the interaction of AcceECN with commonly used variants of TCP, whether standardised or not. Section 5 summarises the features and properties of AcceECN.

Section 6 summarises the protocol fields and numbers that IANA will need to assign and Section 7 points to the aspects of the protocol that will be of interest to the security community.

Appendix A gives pseudocode examples for the various algorithms that AcceECN uses.

### 1.2. Goals

[RFC7560] enumerates requirements that a candidate feedback scheme will need to satisfy, under the headings: resilience, timeliness, integrity, accuracy (including ordering and lack of bias), complexity, overhead and compatibility (both backward and forward). It recognises that a perfect scheme that fully satisfies all the requirements is unlikely and trade-offs between requirements are likely. Section 5 presents the properties of AcceECN against these requirements and discusses the trade-offs made.

The requirements document recognises that a protocol as ubiquitous as TCP needs to be able to serve as-yet-unspecified requirements. Therefore an AccECN receiver aims to act as a generic (dumb) reflector of congestion information so that in future new sender behaviours can be deployed unilaterally.

### 1.3. Experiment Goals

TCP is critical to the robust functioning of the Internet, therefore any proposed modifications to TCP need to be thoroughly tested. The present specification describes an experimental protocol that adds more accurate ECN feedback to the TCP protocol. The intention is to specify the protocol sufficiently so that more than one implementation can be built in order to test its function, robustness and interoperability (with itself and with previous version of ECN and TCP).

The experimental protocol will be considered successful if it satisfies the requirements of [RFC7560] in the consensus opinion of the IETF tcpm working group. In short, this requires that it improves the accuracy and timeliness of TCP's ECN feedback, as claimed in Section 5, while striking a balance between the conflicting requirements of resilience, integrity and minimisation of overhead. It also requires that it is not unduly complex, and that it is compatible with prevalent equipment behaviours in the current Internet, whether or not they comply with standards.

### 1.4. Terminology

AccECN: The more accurate ECN feedback scheme will be called AccECN for short.

Classic ECN: the ECN protocol specified in [RFC3168].

Classic ECN feedback: the feedback aspect of the ECN protocol specified in [RFC3168], including generation, encoding, transmission and decoding of feedback, but not the Data Sender's subsequent response to that feedback.

ACK: A TCP acknowledgement, with or without a data payload.

Pure ACK: A TCP acknowledgement without a data payload.

TCP client: The TCP stack that originates a connection.

TCP server: The TCP stack that responds to a connection request.

**Data Receiver:** The endpoint of a TCP half-connection that receives data and sends AccECN feedback.

**Data Sender:** The endpoint of a TCP half-connection that sends data and receives AccECN feedback.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

### 1.5. Recap of Existing ECN feedback in IP/TCP

ECN [RFC3168] uses two bits in the IP header. Once ECN has been negotiated with the receiver at the transport layer, an ECN sender can set two possible codepoints (ECT(0) or ECT(1)) in the IP header to indicate an ECN-capable transport (ECT). If both ECN bits are zero, the packet is considered to have been sent by a Not-ECN-capable Transport (Not-ECT). When a network node experiences congestion, it will occasionally either drop or mark a packet, with the choice depending on the packet's ECN codepoint. If the codepoint is Not-ECT, only drop is appropriate. If the codepoint is ECT(0) or ECT(1), the node can mark the packet by setting both ECN bits, which is termed 'Congestion Experienced' (CE), or loosely a 'congestion mark'. Table 1 summarises these codepoints.

IP-ECN codepoint (binary)	Codepoint name	Description
00	Not-ECT	Not ECN-Capable Transport
01	ECT(1)	ECN-Capable Transport (1)
10	ECT(0)	ECN-Capable Transport (0)
11	CE	Congestion Experienced

Table 1: The ECN Field in the IP Header

In the TCP header the first two bits in byte 14 are defined as flags for the use of ECN (CWR and ECE in Figure 1 [RFC3168]). A TCP client indicates it supports ECN by setting ECE=CWR=1 in the SYN, and an ECN-enabled server confirms ECN support by setting ECE=1 and CWR=0 in the SYN/ACK. On reception of a CE-marked packet at the IP layer, the Data Receiver starts to set the Echo Congestion Experienced (ECE) flag continuously in the TCP header of ACKs, which ensures the signal is received reliably even if ACKs are lost. The TCP sender confirms that it has received at least one ECE signal by responding with the congestion window reduced (CWR) flag, which allows the TCP receiver to stop repeating the ECN-Echo flag. This always leads to a full RTT

of ACKs with ECE set. Thus any additional CE markings arriving within this RTT cannot be fed back.

The ECN Nonce [RFC3540] is an optional experimental addition to ECN that the TCP sender can use to protect against accidental or malicious concealment of marked or dropped packets. The sender can send an ECN nonce, which is a continuous pseudo-random pattern of ECT(0) and ECT(1) codepoints in the ECN field. The receiver is required to feed back a 1-bit nonce sum that counts the occurrence of ECT(1) packets using the last bit of byte 13 in the TCP header, which is defined as the Nonce Sum (NS) flag.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Header Length				Reserved		N	C	E	U	A	P	R	S	F	
						S	W	C	R	C	S	S	Y	I	
							R	E	G	K	H	T	N	N	

Figure 1: The (post-ECN Nonce) definition of the TCP header flags

## 2. AcceCN Protocol Overview and Rationale

This section provides an informative overview of the AcceCN protocol that will be normatively specified in Section 3

Like the original TCP approach, the Data Receiver of each TCP half-connection sends AcceCN feedback to the Data Sender on TCP acknowledgements, reusing data packets of the other half-connection whenever possible.

The AcceCN protocol has had to be designed in two parts:

- o an essential part that re-uses ECN TCP header bits to feed back the number of arriving CE marked packets. This provides more accuracy than classic ECN feedback, but limited resilience against ACK loss;
- o a supplementary part using a new AcceCN TCP Option that provides additional feedback on the number of bytes that arrive marked with each of the three ECN codepoints (not just CE marks). This provides greater resilience against ACK loss than the essential feedback, but it is more likely to suffer from middlebox interference.

The two part design was necessary, given limitations on the space available for TCP options and given the possibility that certain incorrectly designed middleboxes prevent TCP using any new options.

The essential part overloads the previous definition of the three flags in the TCP header that had been assigned for use by ECN. This design choice deliberately replaces the classic ECN feedback protocol, rather than leaving classic ECN feedback intact and adding more accurate feedback separately because:

- o this efficiently reuses scarce TCP header space, given TCP option space is approaching saturation;
- o a single upgrade path for the TCP protocol is preferable to a fork in the design;
- o otherwise classic and accurate ECN feedback could give conflicting feedback on the same segment, which could open up new security concerns and make implementations unnecessarily complex;
- o middleboxes are more likely to faithfully forward the TCP ECN flags than newly defined areas of the TCP header.

AcceCN is designed to work even if the supplementary part is removed or zeroed out, as long as the essential part gets through.

## 2.1. Capability Negotiation

AcceCN is a change to the wire protocol of the main TCP header, therefore it can only be used if both endpoints have been upgraded to understand it. The TCP client signals support for AcceCN on the initial SYN of a connection and the TCP server signals whether it supports AcceCN on the SYN/ACK. The TCP flags on the SYN that the client uses to signal AcceCN support have been carefully chosen so that a TCP server will interpret them as a request to support the most recent variant of ECN feedback that it supports. Then the client falls back to the same variant of ECN feedback.

An AcceCN TCP client does not send the new AcceCN Option on the SYN as SYN option space is limited and successful negotiation using the flags in the main header is taken as sufficient evidence that both ends also support the AcceCN Option. The TCP server sends the AcceCN Option on the SYN/ACK and the client sends it on the first ACK to test whether the network path forwards the option correctly.

## 2.2. Feedback Mechanism

A Data Receiver maintains four counters initialised at the start of the half-connection. Three count the number of arriving payload bytes marked CE, ECT(1) and ECT(0) respectively. The fourth counts the number of packets arriving marked with a CE codepoint (including control packets without payload if they are CE-marked).

The Data Sender maintains four equivalent counters for the half connection, and the AccECN protocol is designed to ensure they will match the values in the Data Receiver's counters, albeit after a little delay.

Each ACK carries the three least significant bits (LSBs) of the packet-based CE counter using the ECN bits in the TCP header, now renamed the Accurate ECN (ACE) field. The LSBs of each of the three byte counters are carried in the AccECN Option.

### 2.3. Delayed ACKs and Resilience Against ACK Loss

With both the ACE and the AccECN Option mechanisms, the Data Receiver continually repeats the current LSBs of each of its respective counters. Then, even if some ACKs are lost, the Data Sender should be able to infer how much to increment its own counters, even if the protocol field has wrapped.

The 3-bit ACE field can wrap fairly frequently. Therefore, even if it appears to have incremented by one (say), the field might have actually cycled completely then incremented by one. The Data Receiver is required not to delay sending an ACK to such an extent that the ACE field would cycle. However cycling is still a possibility at the Data Sender because a whole sequence of ACKs carrying intervening values of the field might all be lost or delayed in transit.

The fields in the AccECN Option are larger, but they will increment in larger steps because they count bytes not packets. Nonetheless, their size has been chosen such that a whole cycle of the field would never occur between ACKs unless there had been an infeasibly long sequence of ACK losses. Therefore, as long as the AccECN Option is available, it can be treated as a dependable feedback channel.

If the AccECN Option is not available, e.g. it is being stripped by a middlebox, the AccECN protocol will only feed back information on CE markings (using the ACE field). Although not ideal, this will be sufficient, because it is envisaged that neither ECT(0) nor ECT(1) will ever indicate more severe congestion than CE, even though future uses for ECT(0) or ECT(1) are still unclear. Because the 3-bit ACE field is so small, when it is the only field available the Data Sender has to interpret it conservatively assuming the worst possible wrap.

Certain specified events trigger the Data Receiver to include an AccECN Option on an ACK. The rules are designed to ensure that the order in which different markings arrive at the receiver is communicated to the sender (as long as there is no ACK loss).

Implementations are encouraged to send an AcceCN Option more frequently, but this is left up to the implementer.

#### 2.4. Feedback Metrics

The CE packet counter in the ACE field and the CE byte counter in the AcceCN Option both provide feedback on received CE-marks. The CE packet counter includes control packets that do not have payload data, while the CE byte counter solely includes marked payload bytes. If both are present, the byte counter in the option will provide the more accurate information needed for modern congestion control and policing schemes, such as DCTCP or ConEx. If the option is stripped, a simple algorithm to estimate the number of marked bytes from the ACE field is given in Appendix A.3.

Feedback in bytes is recommended in order to protect against the receiver using attacks similar to 'ACK-Division' to artificially inflate the congestion window, which is why [RFC5681] now recommends that TCP counts acknowledged bytes not packets.

#### 2.5. Generic (Dumb) Reflector

The ACE field provides information about CE markings on both data and control packets. According to [RFC3168] the Data Sender is meant to set control packets to Not-ECT. However, mechanisms in certain private networks (e.g. data centres) set control packets to be ECN capable because they are precisely the packets that performance depends on most.

For this reason, AcceCN is designed to be a generic reflector of whatever ECN markings it sees, whether or not they are compliant with a current standard. Then as standards evolve, Data Senders can upgrade unilaterally without any need for receivers to upgrade too. It is also useful to be able to rely on generic reflection behaviour when senders need to test for unexpected interference with markings (for instance [I-D.kuehlewind-tcpm-ecn-fallback] and [I-D.moncaster-tcpm-rcv-cheat]).

The initial SYN is the most critical control packet, so AcceCN provides feedback on whether it is CE marked, even though it is not allowed to be ECN-capable according to RFC 3168. However, middleboxes have been known to overwrite the ECN IP field as if it is still part of the old Type of Service (ToS) field. If a TCP client has set the SYN to Not-ECT, but receives CE feedback, it can detect such middlebox interference and send Not-ECT for the rest of the connection (see [I-D.kuehlewind-tcpm-ecn-fallback] for the detailed fall-back behaviour).

Today, if a TCP server receives CE on a SYN, it cannot know whether it is invalid (or valid) because only the TCP client knows whether it originally marked the SYN as Not-ECT (or ECT). Therefore, the server's only safe course of action is to disable ECN for the connection. Instead, the AccECN protocol allows the server to feed back the CE marking to the client, which then has all the information to decide whether the connection has to fall-back from supporting ECN (or not).

Providing feedback of CE marking on the SYN also supports future scenarios in which SYNs might be ECN-enabled (without prejudging whether they ought to be). For instance, in certain environments such as data centres, it might be appropriate to allow ECN-capable SYNs. Then, if feedback showed the SYN had been CE marked, the TCP client could reduce its initial window (IW). It could also reduce IW conservatively if feedback showed the receiver did not support ECN (because if there had been a CE marking, the receiver would not have understood it). Note that this text merely motivates dumb reflection of CE on a SYN, it does not judge whether a SYN ought to be ECN-capable.

### 3. AccECN Protocol Specification

#### 3.1. Negotiation during the TCP handshake

During the TCP handshake at the start of a connection, to request more accurate ECN feedback the TCP client (host A) MUST set the TCP flags NS=1, CWR=1 and ECE=1 in the initial SYN segment.

If a TCP server (B) that is AccECN enabled receives a SYN with the above three flags set, it MUST set both its half connections into AccECN mode. Then it MUST set the flags CWR=1 and ECE=0 on its response in the SYN/ACK segment to confirm that it supports AccECN. The TCP server MUST NOT set this combination of flags unless the preceding SYN requested support for AccECN as above.

A TCP server in AccECN mode MUST additionally set the flag NS=1 on the SYN/ACK if the SYN was CE-marked (see Section 2.5). If the received SYN was Not-ECT, ECT(0) or ECT(1), it MUST clear NS (NS=0) on the SYN/ACK.

Once a TCP client (A) has sent the above SYN to declare that it supports AccECN, and once it has received the above SYN/ACK segment that confirms that the TCP server supports AccECN, the TCP client MUST set both its half connections into AccECN mode.

If after the normal TCP timeout the TCP client has not received a SYN/ACK to acknowledge its SYN, the SYN might just have been lost,

e.g. due to congestion, or a middlebox might be blocking segments with the AccECN flags. To expedite connection setup, the host SHOULD fall back to NS=CWR=ECE=0 on the retransmission of the SYN. It would make sense to also remove any other experimental fields or options on the SYN in case a middlebox might be blocking them, although the required behaviour will depend on the specification of the other option(s) and any attempt to co-ordinate fall-back between different modules of the stack. Implementers MAY use other fall-back strategies if they are found to be more effective (e.g. attempting to retransmit a second AccECN segment before fall-back, falling back to classic ECN feedback rather than non-ECN, and/or caching the result of a previous attempt to access the same host while negotiating AccECN).

The fall-back procedure if the TCP server receives no ACK to acknowledge a SYN/ACK that tried to negotiate AccECN is specified in Section 3.2.4.

The three flags set to 1 to indicate AccECN support on the SYN have been carefully chosen to enable natural fall-back to prior stages in the evolution of ECN. Table 2 tabulates all the negotiation possibilities for ECN-related capabilities that involve at least one AccECN-capable host. To compress the width of the table, the headings of the first four columns have been severely abbreviated, as follows:

Ac: More \*Ac\*curate ECN Feedback

N: ECN-\*N\*once [RFC3540]

E: \*E\*CN [RFC3168]

I: Not-ECN (\*I\*mplicit congestion notification using packet drop).

Ac	N	E	I	SYN A->B			SYN/ACK B->A			Feedback Mode
AB				NS	CWR	ECE	NS	CWR	ECE	AccECN
AB				1	1	1	0	1	0	AccECN (CE on SYN)
A	B			1	1	1	1	0	1	classic ECN
A		B		1	1	1	0	0	1	classic ECN
A			B	1	1	1	0	0	0	Not ECN
B	A			0	1	1	0	0	1	classic ECN
B		A		0	1	1	0	0	1	classic ECN
B			A	0	0	0	0	0	0	Not ECN
A			B	1	1	1	1	1	1	Not ECN (broken)
A				1	1	1	0	1	1	Not ECN (see Appx B)
A				1	1	1	1	0	0	Not ECN (see Appx B)

Table 2: ECN capability negotiation between Originator (A) and Responder (B)

Table 2 is divided into blocks each separated by an empty row.

1. The top block shows the case already described where both endpoints support AccECN and how the TCP server (B) indicates congestion feedback.
2. The second block shows the cases where the TCP client (A) supports AccECN but the TCP server (B) supports some earlier variant of TCP feedback, indicated in its SYN/ACK. Therefore, as soon as an AccECN-capable TCP client (A) receives the SYN/ACK shown it MUST set both its half connections into the feedback mode shown in the rightmost column.
3. The third block shows the cases where the TCP server (B) supports AccECN but the TCP client (A) supports some earlier variant of TCP feedback, indicated in its SYN. Therefore, as soon as an AccECN-enabled TCP server (B) receives the SYN shown, it MUST set both its half connections into the feedback mode shown in the rightmost column.
4. The fourth block displays combinations that are not valid or currently unused and therefore both ends MUST fall-back to Not ECN for both half connections. Especially the first case (marked 'broken') where all bits set in the SYN are reflected by the receiver in the SYN/ACK, which happens quite often if the TCP

connection is proxied. {ToDo: Consider using the last two cases for AccECN f/b of ECT(0) and ECT(1) on the SYN (Appendix B)}

The following exceptional cases need some explanation:

**ECN Nonce:** An AccECN implementation, whether client or server, sender or receiver, does not need to implement the ECN Nonce behaviour [RFC3540]. AccECN is compatible with an alternative ECN feedback integrity approach that does not use up the ECT(1) codepoint and can be implemented solely at the sender (see Section 4.3).

**Simultaneous Open:** An originating AccECN Host (A), having sent a SYN with NS=1, CWR=1 and ECE=1, might receive another SYN from host B. Host A MUST then enter the same feedback mode as it would have entered had it been a responding host and received the same SYN. Then host A MUST send the same SYN/ACK as it would have sent had it been a responding host (see the third block above).

### 3.2. AccECN Feedback

Each Data Receiver maintains four counters, `r.cep`, `r.ceb`, `r.e0b` and `r.elb`. The CE packet counter (`r.cep`), counts the number of packets the host receives with the CE code point in the IP ECN field, including CE marks on control packets without data. `r.ceb`, `r.e0b` and `r.elb` count the number of TCP payload bytes in packets marked respectively with the CE, ECT(0) and ECT(1) codepoint in their IP-ECN field. When a host first enters AccECN mode, it initialises its counters to `r.cep = 6`, `r.e0b = 1` and `r.ceb = r.elb = 0` (see Appendix A.5). Non-zero initial values are used to be distinct from cases where the fields are incorrectly zeroed (e.g. by middleboxes).

A host feeds back the CE packet counter using the Accurate ECN (ACE) field, as explained in the next section. And it feeds back all the byte counters using the AccECN TCP Option, as specified in Section 3.2.3. Whenever a host feeds back the value of any counter, it MUST report the most recent value, no matter whether it is in a pure ACK, an ACK with new payload data or a retransmission.

#### 3.2.1. The ACE Field

After AccECN has been negotiated on the SYN and SYN/ACK, both hosts overload the three TCP flags ECE, CWR and NS in the main TCP header as one 3-bit field. Then the field is given a new name, ACE, as shown in Figure 2.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Header Length				Reserved			ACE			U	A	P	R	S	F
										R	C	S	S	Y	I
										G	K	H	T	N	N

Figure 2: Definition of the ACE field within bytes 13 and 14 of the TCP Header (when AcceECN has been negotiated and SYN=0).

The original definition of these three flags in the TCP header, including the addition of support for the ECN Nonce, is shown for comparison in Figure 1. This specification does not rename these three TCP flags, it merely overloads them with another name and definition once an AcceECN connection has been established.

A host **MUST** interpret the ECE, CWR and NS flags as the 3-bit ACE counter on a segment with SYN=0 that it sends or receives if both of its half-connections are set into AcceECN mode having successfully negotiated AcceECN (see Section 3.1). A host **MUST NOT** interpret the 3 flags as a 3-bit ACE field on any segment with SYN=1 (whether ACK is 0 or 1), or if AcceECN negotiation is incomplete or has not succeeded.

Both parts of each of these conditions are equally important. For instance, even if AcceECN negotiation has been successful, the ACE field is not defined on any segments with SYN=1 (e.g. a retransmission of an unacknowledged SYN/ACK, or when both ends send SYN/ACKs after AcceECN support has been successfully negotiated during a simultaneous open).

The ACE field encodes the three least significant bits of the r.cep counter, therefore its initial value will be 0b110 (decimal 6). This non-zero initialization allows a TCP server to use a stateless handshake (see Section 4.1) but still detect from the TCP client's first ACK that the client considers it has successfully negotiated AcceECN. If the SYN/ACK was CE marked, the client **MUST** increase its r.cep counter before it sends its first ACK, therefore the initial value of the ACE field will be 0b111 (decimal 7). These values have deliberately been chosen such that they are distinct from [RFC5562] behaviour, where the TCP client would set ECE on the first ACK as feedback for a CE mark on the SYN/ACK.

If the value of the ACE field on the first segment with SYN=0 in either direction is anything other than 0b110 or 0b111, the Data Receiver **MUST** disable ECN for the remainder of the half-connection by marking all subsequent packets as Not-ECT.

### 3.2.2. Safety against Ambiguity of the ACE Field

If too many CE-marked segments are acknowledged at once, or if a long run of ACKs is lost, the 3-bit counter in the ACE field might have cycled between two ACKs arriving at the Data Sender.

Therefore an AccECN Data Receiver SHOULD immediately send an ACK once 'n' CE marks have arrived since the previous ACK, where 'n' SHOULD be 2 and MUST be no greater than 6.

If the Data Sender has not received AccECN TCP Options to give it more dependable information, and it detects that the ACE field could have cycled under the prevailing conditions, it SHOULD conservatively assume that the counter did cycle. It can detect if the counter could have cycled by using the jump in the acknowledgement number since the last ACK to calculate or estimate how many segments could have been acknowledged. An example algorithm to implement this policy is given in Appendix A.2. An implementer MAY develop an alternative algorithm as long as it satisfies these requirements.

If missing acknowledgement numbers arrive later (reordering) and prove that the counter did not cycle, the Data Sender MAY attempt to neutralise the effect of any action it took based on a conservative assumption that it later found to be incorrect.

### 3.2.3. The AccECN Option

The AccECN Option is defined as shown below in Figure 3. It consists of three 24-bit fields that provide the 24 least significant bits of the r.e0b, r.ceb and r.elb counters, respectively. The initial 'E' of each field name stands for 'Echo'.

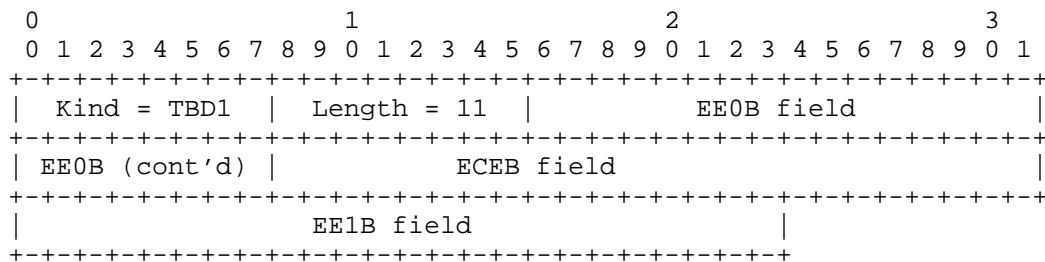


Figure 3: The AccECN Option

The Data Receiver MUST set the Kind field to TBD1, which is registered in Section 6 as a new TCP option Kind called AccECN. An experimental TCP option with Kind=254 MAY be used for initial experiments, with magic number 0xACCE.

Appendix A.1 gives an example algorithm for the Data Receiver to encode its byte counters into the AcceCN Option, and for the Data Sender to decode the AcceCN Option fields into its byte counters.

Note that there is no field to feedback Not-ECT bytes. Nonetheless an algorithm for the Data Sender to calculate the number of payload bytes received as Not-ECT is given in Appendix A.5.

Whenever a Data Receiver sends an AcceCN Option, the rules in Section 3.2.5 expect it to always send a full-length option. To cope with option space limitations, it can omit unchanged fields from the tail of the option, as long as it preserves the order of the remaining fields and includes any field that has changed. The length field MUST indicate which fields are present as follows:

Length=11: EE0B, ECEB, EE1B

Length=8: EE0B, ECEB

Length=5: EE0B

Length=2: (empty)

The empty option of Length=2 is provided to allow for a case where an AcceCN Option has to be sent (e.g. on the SYN/ACK to test the path), but there is very limited space for the option. For initial experiments, the Length field MUST be 2 greater to accommodate the 16-bit magic number.

All implementations of a Data Sender MUST be able to read in AcceCN Options of any of the above lengths. They MUST ignore an AcceCN Option of any other length.

#### 3.2.4. Path Traversal of the AcceCN Option

An AcceCN host MUST NOT include the AcceCN TCP Option on the SYN. Nonetheless, if the AcceCN negotiation using the ECN flags in the main TCP header (Section 3.1) is successful, it implicitly declares that the endpoints also support the AcceCN TCP Option.

If the TCP client indicated AcceCN support, a TCP server that confirms its support for AcceCN (as described in Section 3.1) SHOULD also include an AcceCN TCP Option in the SYN/ACK. A TCP client that has successfully negotiated AcceCN SHOULD include an AcceCN Option in the first ACK at the end of the 3WSH. However, this first ACK is not delivered reliably, so the TCP client SHOULD also include an AcceCN Option on the first data segment it sends (if it ever sends one). A host need not include an AcceCN Option in any of these three cases if

it has cached knowledge that the packet would be likely to be blocked on the path to the other host if it included an AcceCN Option.

If the TCP client has successfully negotiated AcceCN but does not receive an AcceCN Option on the SYN/ACK, it switches into a mode that assumes that the AcceCN Option is not available for this half connection. Similarly, if the TCP server has successfully negotiated AcceCN but does not receive an AcceCN Option on the first ACK or on the first data segment, it switches into a mode that assumes that the AcceCN Option is not available for this half connection.

While a host is in the mode that assumes the AcceCN Option is not available, it MUST adopt the conservative interpretation of the ACE field discussed in Section 3.2.2. However, it cannot make any assumption about support of the AcceCN Option on the other half connection, so it MUST continue to send the AcceCN Option itself.

If after the normal TCP timeout the TCP server has not received an ACK to acknowledge its SYN/ACK, the SYN/ACK might just have been lost, e.g. due to congestion, or a middlebox might be blocking the AcceCN Option. To expedite connection setup, the host SHOULD fall back to NS=CWR=ECE=0 and no AcceCN Option on the retransmission of the SYN/ACK. Implementers MAY use other fall-back strategies if they are found to be more effective (e.g. retransmitting a SYN/ACK with AcceCN TCP flags but not the AcceCN Option; attempting to retransmit a second AcceCN segment before fall-back (most appropriate during high levels of congestion); or falling back to classic ECN feedback rather than non-ECN).

Similarly, if the TCP client detects that the first data segment it sent was lost, it SHOULD fall back to no AcceCN Option on the retransmission. Again, implementers MAY use other fall-back strategies such as attempting to retransmit a second segment with the AcceCN Option before fall-back, and/or caching the result of previous attempts.

Either host MAY include the AcceCN Option in a subsequent segment to retest whether the AcceCN Option can traverse the path.

Currently the Data Sender is not required to test whether the arriving byte counters in the AcceCN Option have been correctly initialised. This allows different initial values to be used as an additional signalling channel in future. If any inappropriate zeroing of these fields is discovered during testing, this approach will need to be reviewed.

### 3.2.5. Usage of the AccECN TCP Option

The following rules determine when a Data Receiver in AccECN mode sends the AccECN TCP Option, and which fields to include:

**Change-Triggered ACKs:** If an arriving packet increments a different byte counter to that incremented by the previous packet, the Data Receiver SHOULD immediately send an ACK with an AccECN Option, without waiting for the next delayed ACK. Certain offload hardware might not be able to support change-triggered ACKs, but otherwise it is important to keep exceptions to this rule to a minimum so that Data Senders can generally rely on this behaviour;

**Continual Repetition:** Otherwise, if arriving packets continue to increment the same byte counter, the Data Receiver can include an AccECN Option on most or all (delayed) ACKs, but it does not have to. If option space is limited on a particular ACK, the Data Receiver MUST give precedence to SACK information about loss. It SHOULD include an AccECN Option if the r.ccb counter has incremented and it MAY include an AccECN Option if r.ec0b or r.ec1b has incremented;

**Full-Length Options Preferred:** It SHOULD always use full-length AccECN Options. It MAY use shorter AccECN Options if space is limited, but it MUST include the counter(s) that have incremented since the previous AccECN Option and it MUST only truncate fields from the right-hand tail of the option to preserve the order of the remaining fields (see Section 3.2.3);

**Beaconing Full-Length Options:** Nonetheless, it MUST include a full-length AccECN TCP Option on at least three ACKs per RTT, or on all ACKs if there are less than three per RTT (see Appendix A.4 for an example algorithm that satisfies this requirement).

The following example series of arriving marks illustrates when a Data Receiver will emit an ACK if it is using a delayed ACK factor of 2 segments and change-triggered ACKs: 01 -> ACK, 01, 01 -> ACK, 10 -> ACK, 10, 01 -> ACK, 01, 11 -> ACK, 01 -> ACK.

For the avoidance of doubt, the change-triggered ACK mechanism ignores the arrival of a control packet with no payload, because it does not alter any byte counters. The change-triggered ACK approach will lead to some additional ACKs but it feeds back the timing and the order in which ECN marks are received with minimal additional complexity.

**Implementation note:** sending an AccECN Option each time a different counter changes and including a full-length AccECN Option on every

delayed ACK will satisfy the requirements described above and might be the easiest implementation, as long as sufficient space is available in each ACK (in total and in the option space).

Appendix A.3 gives an example algorithm to estimate the number of marked bytes from the ACE field alone, if the AcceECN Option is not available.

If a host has determined that segments with the AcceECN Option always seem to be discarded somewhere along the path, it is no longer obliged to follow the above rules.

### 3.3. AcceECN Compliance by TCP Proxies, Offload Engines and other Middleboxes

A large class of middleboxes split TCP connections. Such a middlebox would be compliant with the AcceECN protocol if the TCP implementation on each side complied with the present AcceECN specification and each side negotiated AcceECN independently of the other side.

Another large class of middleboxes intervene to some degree at the transport layer, but attempts to be transparent (invisible) to the end-to-end connection. A subset of this class of middleboxes attempts to 'normalise' the TCP wire protocol by checking that all values in header fields comply with a rather narrow interpretation of the TCP specifications. To comply with the present AcceECN specification, such a middlebox MUST NOT change the ACE field or the AcceECN Option and it MUST attempt to preserve the timing of each ACK (for example, if it coalesced ACKs it would not be AcceECN-compliant). A middlebox claiming to be transparent at the transport layer MUST forward the AcceECN TCP Option unaltered, whether or not the length value matches one of those specified in Section 3.2.3, and whether or not the initial values of the byte-counter fields are correct. This is because blocking apparently invalid values does not improve security (because AcceECN hosts are required to ignore invalid values anyway), while it prevents the standardised set of values being extended in future (because outdated normalisers would block updated hosts from using the extended AcceECN standard).

Hardware to offload certain TCP processing represents another large class of middleboxes, even though it is often a function of a host's network interface and rarely in its own 'box'. Leeway has been allowed in the present AcceECN specification in the expectation that offload hardware could comply and still serve its function. Nonetheless, such hardware MUST attempt to preserve the timing of each ACK (for example, if it coalesced ACKs it would not be AcceECN-compliant).

#### 4. Interaction with Other TCP Variants

This section is informative, not normative.

##### 4.1. Compatibility with SYN Cookies

A TCP server can use SYN Cookies (see Appendix A of [RFC4987]) to protect itself from SYN flooding attacks. It places minimal commonly used connection state in the SYN/ACK, and deliberately does not hold any state while waiting for the subsequent ACK (e.g. it closes the thread). Therefore it cannot record the fact that it entered AccECN mode for both half-connections. Indeed, it cannot even remember whether it negotiated the use of classic ECN [RFC3168].

Nonetheless, such a server can determine that it negotiated AccECN as follows. If a TCP server using SYN Cookies supports AccECN and if the first ACK it receives contains an ACE field with the value 0b110 or 0b111, it can assume that:

- o the TCP client must have requested AccECN support on the SYN
- o it (the server) must have confirmed that it supported AccECN

Therefore the server can switch itself into AccECN mode, and continue as if it had never forgotten that it switched itself into AccECN mode earlier.

##### 4.2. Compatibility with Other TCP Options and Experiments

AccECN is compatible (at least on paper) with the most commonly used TCP options: MSS, time-stamp, window scaling, SACK and TCP-AO. It is also compatible with the recent promising experimental TCP options TCP Fast Open (TFO [RFC7413]) and Multipath TCP (MPTCP [RFC6824]). AccECN is friendly to all these protocols, because space for TCP options is particularly scarce on the SYN, where AccECN consumes zero additional header space.

When option space is under pressure from other options, Section 3.2.5 provides guidance on how important it is to send an AccECN Option and whether it needs to be a full-length option.

##### 4.3. Compatibility with Feedback Integrity Mechanisms

The ECN Nonce [RFC3540] is an experimental IETF specification intended to allow a sender to test whether ECN CE markings (or losses) introduced in one network are being suppressed by the receiver or anywhere else in the feedback loop, such as another network or a middlebox. The ECN nonce has not been deployed as far

as can be ascertained. The nonce would now be nearly impossible to deploy retrospectively, because to catch a misbehaving receiver it relies on the receiver volunteering feedback information to incriminate itself. A receiver that has been modified to misbehave can simply claim that it does not support nonce feedback, which will seem unremarkable given so many other hosts do not support it either.

With minor changes AcceCN could be optimised for the possibility that the ECT(1) codepoint might be used as a nonce. However, given the nonce is now probably undeployable, the AcceCN design has been generalised so that it ought to be able to support other possible uses of the ECT(1) codepoint, such as a lower severity or a more instant congestion signal than CE.

Three alternative mechanisms are available to assure the integrity of ECN and/or loss signals. AcceCN is compatible with any of these approaches:

- o The Data Sender can test the integrity of the receiver's ECN (or loss) feedback by occasionally setting the IP-ECN field to a value normally only set by the network (and/or deliberately leaving a sequence number gap). Then it can test whether the Data Receiver's feedback faithfully reports what it expects [I-D.moncaster-tcpm-rcv-cheat]. Unlike the ECN Nonce, this approach does not waste the ECT(1) codepoint in the IP header, it does not require standardisation and it does not rely on misbehaving receivers volunteering to reveal feedback information that allows them to be detected. However, setting the CE mark by the sender might conceal actual congestion feedback from the network and should therefore only be done sparsely.
- o Networks generate congestion signals when they are becoming congested, so they are more likely than Data Senders to be concerned about the integrity of the receiver's feedback of these signals. A network can enforce a congestion response to its ECN markings (or packet losses) using congestion exposure (ConEx) audit [I-D.ietf-conex-abstract-mech]. Whether the receiver or a downstream network is suppressing congestion feedback or the sender is unresponsive to the feedback, or both, ConEx audit can neutralise any advantage that any of these three parties would otherwise gain.

ConEx is a change to the Data Sender that is most useful when combined with AcceCN. Without AcceCN, the ConEx behaviour of a Data Sender would have to be more conservative than would be necessary if it had the accurate feedback of AcceCN.

- o The TCP authentication option (TCP-AO [RFC5925]) can be used to detect any tampering with AcceCN feedback between the Data Receiver and the Data Sender (whether malicious or accidental). The AcceCN fields are immutable end-to-end, so they are amenable to TCP-AO protection, which covers TCP options by default. However, TCP-AO is often too brittle to use on many end-to-end paths, where middleboxes can make verification fail in their attempts to improve performance or security, e.g. by resegmentation or shifting the sequence space.

## 5. Protocol Properties

This section is informative not normative. It describes how well the protocol satisfies the agreed requirements for a more accurate ECN feedback protocol [RFC7560].

**Accuracy:** From each ACK, the Data Sender can infer the number of new CE marked segments since the previous ACK. This provides better accuracy on CE feedback than classic ECN. In addition if the AcceCN Option is present (not blocked by the network path) the number of bytes marked with CE, ECT(1) and ECT(0) are provided.

**Overhead:** The AcceCN scheme is divided into two parts. The essential part reuses the 3 flags already assigned to ECN in the IP header. The supplementary part adds an additional TCP option consuming up to 11 bytes. However, no TCP option is consumed in the SYN.

**Ordering:** The order in which marks arrive at the Data Receiver is preserved in AcceCN feedback, because the Data Receiver is expected to send an ACK immediately whenever a different mark arrives.

**Timeliness:** While the same ECN markings are arriving continually at the Data Receiver, it can defer ACKs as TCP does normally, but it will immediately send an ACK as soon as a different ECN marking arrives.

**Timeliness vs Overhead:** Change-Triggered ACKs are intended to enable latency-sensitive uses of ECN feedback by capturing the timing of transitions but not wasting resources while the state of the signalling system is stable. The receiver can control how frequently it sends the AcceCN TCP Option and therefore it can control the overhead induced by AcceCN.

**Resilience:** All information is provided based on counters. Therefore if ACKs are lost, the counters on the first ACK

following the losses allows the Data Sender to immediately recover the number of the ECN markings that it missed.

**Resilience against Bias:** Because feedback is based on repetition of counters, random losses do not remove any information, they only delay it. Therefore, even though some ACKs are change-triggered, random losses will not alter the proportions of the different ECN markings in the feedback.

**Resilience vs Overhead:** If space is limited in some segments (e.g. because more option are need on some segments, such as the SACK option after loss), the Data Receiver can send AccECN Options less frequently or truncate fields that have not changed, usually down to as little as 5 bytes. However, it has to send a full-sized AccECN Option at least three times per RTT, which the Data Sender can rely on as a regular beacon or checkpoint.

**Resilience vs Timeliness and Ordering:** Ordering information and the timing of transitions cannot be communicated in three cases: i) during ACK loss; ii) if something on the path strips the AccECN Option; or iii) if the Data Receiver is unable to support Change-Triggered ACKs.

**Complexity:** An AccECN implementation solely involves simple counter increments, some modulo arithmetic to communicate the least significant bits and allow for wrap, and some heuristics for safety against fields cycling due to prolonged periods of ACK loss. Each host needs to maintain eight additional counters. The hosts have to apply some additional tests to detect tampering by middleboxes, but in general the protocol is simple to understand, simple to implement and requires few cycles per packet to execute.

**Integrity:** AccECN is compatible with at least three approaches that can assure the integrity of ECN feedback. If the AccECN Option is stripped the resolution of the feedback is degraded, but the integrity of this degraded feedback can still be assured.

**Backward Compatibility:** If only one endpoint supports the AccECN scheme, it will fall-back to the most advanced ECN feedback scheme supported by the other end.

**Backward Compatibility:** If the AccECN Option is stripped by a middlebox, AccECN still provides basic congestion feedback in the ACE field. Further, AccECN can be used to detect mangling of the IP ECN field; mangling of the TCP ECN flags; blocking of ECT-marked segments; and blocking of segments carrying the AccECN Option. It can detect these conditions during TCP's 3WSH so that

it can fall back to operation without ECN and/or operation without the AccECN Option.

Forward Compatibility: The behaviour of endpoints and middleboxes is carefully defined for all reserved or currently unused codepoints in the scheme, to ensure that any blocking of anomalous values is always at least under reversible policy control.

## 6. IANA Considerations

This document defines a new TCP option for AccECN, assigned a value of TBD1 (decimal) from the TCP option space. This value is defined as:

Kind	Length	Meaning	Reference
TBD1	N	Accurate ECN (AccECN)	RFC XXXX

[TO BE REMOVED: This registration should take place at the following location: <http://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1>]

Early implementation before the IANA allocation MUST follow [RFC6994] and use experimental option 254 and magic number 0xACCE (16 bits) {ToDo register this with IANA}, then migrate to the new option after the allocation.

## 7. Security Considerations

If ever the supplementary part of AccECN based on the new AccECN TCP Option is unusable (due for example to middlebox interference) the essential part of AccECN's congestion feedback offers only limited resilience to long runs of ACK loss (see Section 3.2.2). These problems are unlikely to be due to malicious intervention (because if an attacker could strip a TCP option or discard a long run of ACKs it could wreak other arbitrary havoc). However, it would be of concern if AccECN's resilience could be indirectly compromised during a flooding attack. AccECN is still considered safe though, because if the option is not presented, the AccECN Data Sender is then required to switch to more conservative assumptions about wrap of congestion indication counters (see Section 3.2.2 and Appendix A.2).

Section 4.1 describes how a TCP server can negotiate AccECN and use the SYN cookie method for mitigating SYN flooding attacks.

There is concern that ECN markings could be altered or suppressed, particularly because a misbehaving Data Receiver could increase its own throughput at the expense of others. Given the experimental ECN nonce is now probably undeployable, AcceCN has been generalised for other possible uses of the ECT(1) codepoint to avoid obsolescence of the codepoint even if the nonce mechanism is obsoleted. AcceCN is compatible with the three other schemes known to assure the integrity of ECN feedback (see Section 4.3 for details). If the AcceCN Option is stripped by an incorrectly implemented middlebox, the resolution of the feedback will be degraded, but the integrity of this degraded information can still be assured.

The AcceCN protocol is not believed to introduce any new privacy concerns, because it merely counts and feeds back signals at the transport layer that had already been visible at the IP layer.

## 8. Acknowledgements

We want to thank Koen De Schepper, Praveen Balasubramanian and Michael Welzl for their input and discussion. The idea of using the three ECN-related TCP flags as one field for more accurate TCP-ECN feedback was first introduced in the re-ECN protocol that was the ancestor of ConEx.

Bob Briscoe was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700) and through the Trilogy 2 project (ICT-317756). The views expressed here are solely those of the authors.

## 9. Comments Solicited

Comments and questions are encouraged and very welcome. They can be addressed to the IETF TCP maintenance and minor modifications working group mailing list <tcpm@ietf.org>, and/or to the authors.

## 10. References

### 10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <<http://www.rfc-editor.org/info/rfc3168>>.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <<http://www.rfc-editor.org/info/rfc5681>>.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <<http://www.rfc-editor.org/info/rfc6994>>.

## 10.2. Informative References

- [I-D.bensley-tcpm-dctcp] Bensley, S., Eggert, L., Thaler, D., Balasubramanian, P., and G. Judd, "Microsoft's Datacenter TCP (DCTCP): TCP Congestion Control for Datacenters", draft-bensley-tcpm-dctcp-05 (work in progress), July 2015.
- [I-D.ietf-conex-abstract-mech] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism and Requirements", draft-ietf-conex-abstract-mech-13 (work in progress), October 2014.
- [I-D.kuehlewind-tcpm-ecn-fallback] Kuehlewind, M. and B. Trammell, "A Mechanism for ECN Path Probing and Fallback", draft-kuehlewind-tcpm-ecn-fallback-01 (work in progress), September 2013.
- [I-D.moncaster-tcpm-rcv-cheat] Moncaster, T., Briscoe, B., and A. Jacquet, "A TCP Test to Allow Senders to Identify Receiver Non-Compliance", draft-moncaster-tcpm-rcv-cheat-03 (work in progress), July 2014.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <<http://www.rfc-editor.org/info/rfc3540>>.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <<http://www.rfc-editor.org/info/rfc4987>>.

- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <<http://www.rfc-editor.org/info/rfc5562>>.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <<http://www.rfc-editor.org/info/rfc5925>>.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <<http://www.rfc-editor.org/info/rfc6824>>.
- [RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <<http://www.rfc-editor.org/info/rfc7413>>.
- [RFC7560] Kuehlewind, M., Ed., Scheffenegger, R., and B. Briscoe, "Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback", RFC 7560, DOI 10.17487/RFC7560, August 2015, <<http://www.rfc-editor.org/info/rfc7560>>.

## Appendix A. Example Algorithms

This appendix is informative, not normative. It gives example algorithms that would satisfy the normative requirements of the AcceCN protocol. However, implementers are free to choose other ways to implement the requirements.

### A.1. Example Algorithm to Encode/Decode the AcceCN Option

The example algorithms below show how a Data Receiver in AcceCN mode could encode its CE byte counter `r.ceb` into the ECEB field within the AcceCN TCP Option, and how a Data Sender in AcceCN mode could decode the ECEB field into its byte counter `s.ceb`. The other counters for bytes marked ECT(0) and ECT(1) in the AcceCN Option would be similarly encoded and decoded.

It is assumed that each local byte counter is an unsigned integer greater than 24b (probably 32b), and that the following constant has been assigned:

$$\text{DIVOPT} = 2^{24}$$

Every time a CE marked data segment arrives, the Data Receiver increments its local value of `r.ceb` by the size of the TCP Data. Whenever it sends an ACK with the AcceCN Option, the value it writes into the ECEB field is

$$\text{ECEB} = \text{r.ceb} \% \text{DIVOPT}$$

where `'%'` is the modulo operator.

On the arrival of an AcceCN Option, the Data Sender uses the TCP acknowledgement number and any SACK options to calculate `newlyAckedB`, the amount of new data that the ACK acknowledges in bytes. If `newlyAckedB` is negative it means that a more up to date ACK has already been processed, so this ACK has been superseded and the Data Sender has to ignore the AcceCN Option. Then the Data Sender calculates the minimum difference `d.ceb` between the ECEB field and its local `s.ceb` counter, using modulo arithmetic as follows:

```
if (newlyAckedB >= 0) {
    d.ceb = (ECEB + DIVOPT - (s.ceb % DIVOPT)) % DIVOPT
    s.ceb += d.ceb
}
```

For example, if `s.ceb` is 33,554,433 and ECEB is 1461 (both decimal), then

```
s.ceb % DIVOPT = 1
d.ceb = (1461 + 2^24 - 1) % 2^24
      = 1460
s.ceb = 33,554,433 + 1460
      = 33,555,893
```

#### A.2. Example Algorithm for Safety Against Long Sequences of ACK Loss

The example algorithms below show how a Data Receiver in AcceCN mode could encode its CE packet counter `r.cep` into the ACE field, and how the Data Sender in AcceCN mode could decode the ACE field into its `s.cep` counter. The Data Sender's algorithm includes code to heuristically detect a long enough unbroken string of ACK losses that could have concealed a cycle of the congestion counter in the ACE field of the next ACK to arrive.

Two variants of the algorithm are given: i) a more conservative variant for a Data Sender to use if it detects that the AcceCN Option is not available (see Section 3.2.2 and Section 3.2.4); and ii) a less conservative variant that is feasible when complementary information is available from the AcceCN Option.

##### A.2.1. Safety Algorithm without the AcceCN Option

It is assumed that each local packet counter is a sufficiently sized unsigned integer (probably 32b) and that the following constant has been assigned:

```
DIVACE = 2^3
```

Every time a CE marked packet arrives, the Data Receiver increments its local value of `r.cep` by 1. It repeats the same value of ACE in every subsequent ACK until the next CE marking arrives, where

```
ACE = r.cep % DIVACE.
```

If the Data Sender received an earlier value of the counter that had been delayed due to ACK reordering, it might incorrectly calculate that the ACE field had wrapped. Therefore, on the arrival of every ACK, the Data Sender uses the TCP acknowledgement number and any SACK options to calculate `newlyAckdB`, the amount of new data that the ACK acknowledges. If `newlyAckdB` is negative it means that a more up to date ACK has already been processed, so this ACK has been superseded and the Data Sender has to ignore the AcceCN Option. If `newlyAckdB` is zero, to break the tie the Data Sender could use timestamps (if present) to work out `newlyAckdT`, the amount of new time that the ACK acknowledges. Then the Data Sender calculates the minimum difference

d.cep between the ACE field and its local s.cep counter, using modulo arithmetic as follows:

```
if ((newlyAcedB > 0) || (newlyAcedB == 0 && newlyAcedT > 0))
    d.cep = (ACE + DIVACE - (s.cep % DIVACE)) % DIVACE
```

Section 3.2.2 requires the Data Sender to assume that the ACE field did cycle if it could have cycled under prevailing conditions. The 3-bit ACE field in an arriving ACK could have cycled and become ambiguous to the Data Sender if a row of ACKs goes missing that covers a stream of data long enough to contain 8 or more CE marks. We use the word 'missing' rather than 'lost', because some or all the missing ACKs might arrive eventually, but out of order. Even if some of the lost ACKs are piggy-backed on data (i.e. not pure ACKs) retransmissions will not repair the lost AcceCN information, because AcceCN requires retransmissions to carry the latest AcceCN counters, not the original ones.

The phrase 'under prevailing conditions' allows the Data Sender to take account of the prevailing size of data segments and the prevailing CE marking rate just before the sequence of ACK losses. However, we shall start with the simplest algorithm, which assumes segments are all full-sized and ultra-conservatively it assumes that ECN marking was 100% on the forward path when ACKs on the reverse path started to all be dropped. Specifically, if newlyAcedB is the amount of data that an ACK acknowledges since the previous ACK, then the Data Sender could assume that this acknowledges newlyAcedPkt full-sized segments, where newlyAcedPkt = newlyAcedB/MSS. Then it could assume that the ACE field incremented by

```
dSafer.cep = newlyAcedPkt - ((newlyAcedPkt - d.cep) % DIVACE),
```

For example, imagine an ACK acknowledges newlyAcedPkt=9 more full-size segments than any previous ACK, and that ACE increments by a minimum of 2 CE marks (d.cep=2). The above formula works out that it would still be safe to assume 2 CE marks (because  $9 - ((9-2) \% 8) = 2$ ). However, if ACE increases by a minimum of 2 but acknowledges 10 full-sized segments, then it would be necessary to assume that there could have been 10 CE marks (because  $10 - ((10-2) \% 8) = 10$ ).

Implementers could build in more heuristics to estimate prevailing average segment size and prevailing ECN marking. For instance, newlyAcedPkt in the above formula could be replaced with newlyAcedPktHeur = newlyAcedPkt\*p\*MSS/s, where s is the prevailing segment size and p is the prevailing ECN marking probability. However, ultimately, if TCP's ECN feedback becomes inaccurate it still has loss detection to fall back on. Therefore, it would seem safe to implement a simple algorithm, rather than a perfect one.

The simple algorithm for dSafer.cep above requires no monitoring of prevailing conditions and it would still be safe if, for example, segments were on average at least 5% of full-sized as long as ECN marking was 5% or less. Assuming it was used, the Data Sender would increment its packet counter as follows:

```
s.cep += dSafer.cep
```

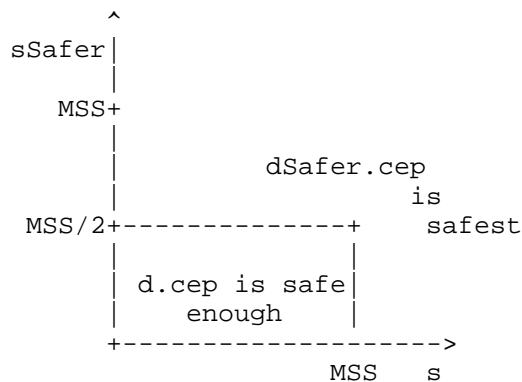
If missing acknowledgement numbers arrive later (due to reordering), Section 3.2.2 says "the Data Sender MAY attempt to neutralise the effect of any action it took based on a conservative assumption that it later found to be incorrect". To do this, the Data Sender would have to store the values of all the relevant variables whenever it made assumptions, so that it could re-evaluate them later. Given this could become complex and it is not required, we do not attempt to provide an example of how to do this.

#### A.2.2. Safety Algorithm with the AcceCN Option

When the AcceCN Option is available on the ACKs before and after the possible sequence of ACK losses, if the Data Sender only needs CE-marked bytes, it will have sufficient information in the AcceCN Option without needing to process the ACE field. However, if for some reason it needs CE-marked packets, if dSafer.cep is different from d.cep, it can calculate the average marked segment size that each implies to determine whether d.cep is likely to be a safe enough estimate. Specifically, it could use the following algorithm, where d.ceb is the amount of newly CE-marked bytes (see Appendix A.1):

```
SAFETY_FACTOR = 2
if (dSafer.cep > d.cep) {
    s = d.ceb/d.cep
    if (s <= MSS) {
        sSafer = d.ceb/dSafer.cep
        if (sSafer < MSS/SAFETY_FACTOR)
            dSafer.cep = d.cep      % d.cep is a safe enough estimate
    } % else
        % No need for else; dSafer.cep is already correct,
        % because d.cep must have been too small
}
```

The chart below shows when the above algorithm will consider d.cep can replace dSafer.cep as a safe enough estimate of the number of CE-marked packets:



The following examples give the reasoning behind the algorithm, assuming  $MSS=1,460$  [B]:

- o if  $d.cep=0$ ,  $dSafer.cep=8$  and  $d.ceb=1,460$ , then  $s=infinity$  and  $sSafer=182.5$ .  
Therefore even though the average size of 8 data segments is unlikely to have been as small as  $MSS/8$ ,  $d.cep$  cannot have been correct, because it would imply an average segment size greater than the  $MSS$ .
- o if  $d.cep=2$ ,  $dSafer.cep=10$  and  $d.ceb=1,460$ , then  $s=730$  and  $sSafer=146$ .  
Therefore  $d.cep$  is safe enough, because the average size of 10 data segments is unlikely to have been as small as  $MSS/10$ .
- o if  $d.cep=7$ ,  $dSafer.cep=15$  and  $d.ceb=10,200$ , then  $s=1,457$  and  $sSafer=680$ .  
Therefore  $d.cep$  is safe enough, because the average data segment size is more likely to have been just less than one  $MSS$ , rather than below  $MSS/2$ .

If pure ACKs were allowed to be ECN-capable, missing ACKs would be far less likely. However, because [RFC3168] currently precludes this, the above algorithm assumes that pure ACKs are not ECN-capable.

#### A.3. Example Algorithm to Estimate Marked Bytes from Marked Packets

If the AccECN Option is not available, the Data Sender can only decode CE-marking from the ACE field in packets. Every time an ACK arrives, to convert this into an estimate of CE-marked bytes, it needs an average of the segment size,  $s_{ave}$ . Then it can add or subtract  $s_{ave}$  from the value of  $d.ceb$  as the value of  $d.cep$  increments or decrements.

To calculate `s_ave`, it could keep a record of the byte numbers of all the boundaries between packets in flight (including control packets), and recalculate `s_ave` on every ACK. However it would be simpler to merely maintain a counter `packets_in_flight` for the number of packets in flight (including control packets), which it could update once per RTT. Either way, it would estimate `s_ave` as:

$$s\_ave \sim \text{flightsize} / \text{packets\_in\_flight},$$

where `flightsize` is the variable that TCP already maintains for the number of bytes in flight. To avoid floating point arithmetic, it could right-bit-shift by `lg(packets_in_flight)`, where `lg()` means log base 2.

An alternative would be to maintain an exponentially weighted moving average (EWMA) of the segment size:

$$s\_ave = a * s + (1-a) * s\_ave,$$

where `a` is the decay constant for the EWMA. However, then it is necessary to choose a good value for this constant, which ought to depend on the number of packets in flight. Also the decay constant needs to be power of two to avoid floating point arithmetic.

#### A.4. Example Algorithm to Beacon AccECN Options

Section 3.2.5 requires a Data Receiver to beacon a full-length AccECN Option at least 3 times per RTT. This could be implemented by maintaining a variable to store the number of ACKs (pure and data ACKs) since a full AccECN Option was last sent and another for the approximate number of ACKs sent in the last round trip time:

```
if (acks_since_full_last_sent > acks_in_round / BEACON_FREQ)
    send_full_AccECN_Option()
```

For optimised integer arithmetic, `BEACON_FREQ = 4` could be used, rather than 3, so that the division could be implemented as an integer right bit-shift by `lg(BEACON_FREQ)`.

In certain operating systems, it might be too complex to maintain `acks_in_round`. In others it might be possible by tagging each data segment in the retransmit buffer with the number of ACKs sent at the point that segment was sent. This would not work well if the Data Receiver was not sending data itself, in which case it might be necessary to beacon based on time instead, as follows:

```
if (time_now > time_last_option_sent + RTT / BEACON_FREQ)
    send_full_AccECN_Option()
```

However, this time-based approach does not work well when all the ACKs are sent early in each round trip, as is the case during slow-start.

{ToDo: A simple and robust beaconing algorithm for all circumstances is still work-in-progress.}

#### A.5. Example Algorithm to Count Not-ECT Bytes

A Data Sender in AccECN mode can infer the amount of TCP payload data arriving at the receiver marked Not-ECT from the difference between the amount of newly ACKed data and the sum of the bytes with the other three markings, d.ceb, d.e0b and d.elb. Note that, because r.e0b is initialised to 1 and the other two counters are initialised to 0, the initial sum will be 1, which matches the initial offset of the TCP sequence number on completion of the 3WHS.

For this approach to be precise, it has to be assumed that spurious (unnecessary) retransmissions do not lead to double counting. This assumption is currently correct, given that RFC 3168 requires that the Data Sender marks retransmitted segments as Not-ECT. However, the converse is not true; necessary transmissions will result in under-counting.

However, such precision is unlikely to be necessary. The only known use of a count of Not-ECT marked bytes is to test whether equipment on the path is clearing the ECN field (perhaps due to an out-dated attempt to clear, or bleach, what used to be the ToS field). To detect bleaching it will be sufficient to detect whether nearly all bytes arrive marked as Not-ECT. Therefore there should be no need to keep track of the details of retransmissions.

#### Appendix B. Alternative Design Choices (To Be Removed Before Publication)

This appendix is informative, not normative. It records alternative designs that the authors chose not to include in the normative specification, but which the IETF might wish to consider for inclusion:

Feedback all four ECN codepoints on the SYN/ACK: The last two negotiation combinations in Table 2 could also be used to indicate AccECN support and to feedback that the arriving SYN was ECT(0) or ECT(1). This could be used to probe the client to server path for incorrect forwarding of the ECN field [I-D.kuehlewind-tcpm-ecn-fallback]. Note, however, that it would be unremarkable if ECN on the SYN was zeroed by security devices,

given RFC 3168 prohibited ECT on SYN because it enables DoS attacks.

Feedback all four ECN codepoints on the First ACK: To probe the server to client path for incorrect ECN forwarding, it could be useful to have four feedback states on the first ACK from the TCP client. This could be achieved by assigning four combinations of the ECN flags in the main TCP header, and only initialising the ACE field on subsequent segments.

Empty AceECN Option: It might be useful to allow an empty (Length=2) AceECN Option on the SYN/ACK and first ACK. Then if a host had to omit the option because there was insufficient space for a larger option, it would not give the impression to the other end that a middlebox had stripped the option.

#### Appendix C. Open Protocol Design Issues (To Be Removed Before Publication)

1. Currently it is specified that the receiver 'SHOULD' use Change-Triggered ACKs. It is controversial whether this ought to be a 'MUST' instead. A 'SHOULD' would leave the Data Sender uncertain whether it can rely on the timing and ordering information in ACKs. If the sender guesses wrongly, it will probably introduce at least 1RTT of delay before it can use this timing information. Ironically it will most likely be wanting this information to reduce ramp-up delay. A 'MUST' could make it hard to implement AceECN in offload hardware. However, it is not known whether AceECN would be hard to implement in such hardware even with a 'SHOULD' here. For instance, was it hard to offload DCTCP to hardware because of change-triggered ACKs, or was this just one of many reasons? The choice between MUST and SHOULD here is critical. Before that choice is made, a clear use-case for certainty of timing and ordering information is needed, plus well-informed discussion about hardware offload constraints.
2. There is possibly a concern that a receiver could deliberately omit the AceECN Option pretending that it had been stripped by a middlebox. No known way can yet be contrived to take advantage of this downgrade attack, but it is mentioned here in case someone else can contrive one.
3. The s.cep counter might increase even if the s.ceb counter does not (e.g. due to a CE-marked control packet). The sender's response to such a situation is considered out of scope, because this ought to be dealt with in whatever future specification allows ECN-capable control packets. However, it is possible that the situation might arise even if the sender has not sent ECN-

capable control packets, in which case, this draft might need to give some advice on how the sender should respond.

#### Appendix D. Changes in This Version (To Be Removed Before Publication)

The difference between any pair of versions can be displayed at  
<<http://datatracker.ietf.org/doc/draft-kuehlewind-tcpm-accurate-ecn/history/>>

From 04 to 05::

- \* Corrected ambiguity between Classic ECN and Classic ECN feedback throughout
- \* Changed MUST to SHOULD send AcceECN option on SYN/ACK last ACK of 3WHS and first data segment from client, to allow for cached knowledge of option traversal problems.
- \* Removed duplication of normative language about sending a full-length option in the sections on "The AcceECN Option" and "Usage of the AcceECN Option", and mutually cross referenced.
- \* Acknowledged Koen De Schepper and Praveen Balasubramanian
- \* Noted in Appendix that algo to beacon a full-length option is work-in-progress
- \* Editorial corrections and clarifications throughout

#### Authors' Addresses

Bob Briscoe  
Simula Research Laboratory  
  
EMail: [ietf@bobbriscoe.net](mailto:ietf@bobbriscoe.net)  
URI: <http://bobbriscoe.net/>

Mirja Kuehlewind  
ETH Zurich  
Gloriastrasse 35  
Zurich 8092  
Switzerland  
  
EMail: [mirja.kuehlewind@tik.ee.ethz.ch](mailto:mirja.kuehlewind@tik.ee.ethz.ch)

Richard Scheffenegger  
NetApp, Inc.  
Am Euro Platz 2  
Vienna 1120  
Austria

Phone: +43 1 3676811 3146  
EMail: rs@netapp.com

tcpm  
Internet-Draft  
Intended status: Experimental  
Expires: January 14, 2013

M. Kuehlewind, Ed.  
University of Stuttgart  
R. Scheffenegger  
NetApp, Inc.  
July 13, 2012

Accurate ECN Feedback Option in TCP  
draft-kuehlewind-tcpm-accurate-ecn-option-01

Abstract

This document specifies an TCP option to get accurate Explicit Congestion Notification (ECN) feedback from the receiver. ECN is an IP/TCP mechanism where network nodes can mark IP packets instead of dropping them to indicate congestion to the end-points. An ECN-capable receiver will feedback this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recently new TCP mechanisms like ConEx or DCTCP need more accurate feedback information in the case where more than one marking is received in one RTT. This TCP extension can be used in addition to the classic ECN as well as with a more accurate ECN scheme recently proposed which reuses the ECN bit in the TCP header.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 14, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal

Provisions Relating to IETF Documents  
(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
1.1. Overview ECN and ECN Nonce in IP . . . . .	3
1.2. Requirements Language . . . . .	3
2. Negotiation of Accurate ECN feedback . . . . .	4
3. Accurate ECN (AccECN) feedback Option Specification . . . . .	5
4. Acknowledgements . . . . .	6
5. IANA Considerations . . . . .	6
6. Security Considerations . . . . .	6
7. References . . . . .	6
7.1. Normative References . . . . .	6
7.2. Informative References . . . . .	6
Authors' Addresses . . . . .	7

## 1. Introduction

Explicit Congestion Notification (ECN) [RFC3168] is an IP/TCP mechanism where network nodes can mark IP packets instead of dropping them to indicate congestion to the end-points. An ECN-capable receiver will feedback this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recently proposed mechanisms like Congestion Exposure (ConEx) or DCTCP [Ali10] need more accurate feedback information in case when more than one marking is received in one RTT.

This documents specifies an TCP option to provide more than one ECN feedback signal per RTT. This modification does not obsolete [RFC3168]. This TCP extension can be used in addition to the classic ECN as well as in addition to more accurate ECN scheme recently proposed which reuses the ECN bits in the TCP header for the same purpose than this extension --- more accurate ECN feedback (see [I-D.kuehlewind-conex-accurate-ecn]). Note that a new TCP extension can experience deployment problems by middleboxes dropping unknown options. Thus the ECN feedback in the TCP header is still needed to ensure ECN feedback. Moreover, this option will increase the header length for all kind of TCP packets which can cause additional load in case of severe congestion (on the feedback channel).

### 1.1. Overview ECN and ECN Nonce in IP

ECN requires two bits in the IP header. The ECN capability of a packet is indicated, when either one of the two bits is set. An ECN sender can set one or the other bit to indicate an ECN-capable transport (ECT) which results in two signals --- ECT(0) and respectively ECT(1). A network node can set both bits simultaneously when it experiences congestion. When both bits are set the packets is regarded as "Congestion Experienced" (CE).

ECN-Nonce [RFC3540] is an optional addition to ECN that is used to protects the TCP sender against accidental or malicious concealment of marked or dropped packets. With ECN-Nonce a nonce sum is maintain that counts the occurrence of ECT(1) packets.

### 1.2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

We use the following terminology from [RFC3168] and [RFC3540]:

The ECN field in the IP header:

CE: the Congestion Experienced codepoint; and

ECT(0)/ECT(1): either one of the two ECN-Capable Transport codepoints.

In this document, we will call the ECN feedback scheme as specified in [RFC3168] the 'classic ECN'. A 'congestion mark' is defined as an IP packet where the CE codepoint is set.

## 2. Negotiation of Accurate ECN feedback

As there is only limited space in the TCP Options, particularly during the initial three-way handshake, an abbreviated Option is used to negotiate for Accurate ECN feedback. This option also initiates all counters to an initial value of zero at the receiving side.

TCP Accurate ECN Option Negotiation:

Kind: TBD

Length: 2 bytes

```

+-----+-----+
| Kind | 2 |
+-----+-----+
  1       1

```

Figure 1: Accurate ECN feedback TCP option negotiation

This abbreviated option is only valid in a <SYN> or <SYN,ACK> segment, during a three way handshake. The negotiation follows the same procedure as with other TCP options, i.e. SACK. A TCP sender MAY send the accurate ECN feedback negotiation option in an initial SYN segment and MAY send a more accurate ECN option (see Section 3) in other segments only if it received this option negotiation in the initial <SYN> segment or <SYN,ACK> for the connection. A TCP receiver MAY send an <SYN,ACK> segment with the accurate ECN feedback negotiation option in response to a received accurate ECN feedback negotiation option in the <SYN>. If both ends indicate that they support Accurate ECN (AcceCN) feedback, the AcceCN option SHOULD be used in any subsequent TCP segment. A TCP sender or receiver MUST only negotiate for the AcceCN option if ECN is negotiated as well.

### 3. Accurate ECN (AcceCN) feedback Option Specification

A TCP receiver, that provides Accurate ECN feedback, will maintain a counter for the number of ECT(0), ECT(1), CE, non-ECT marked and lost packets as well as the cumulative number of bytes of CE marked packets. The TCP option to provide the Accurate ECN (AcceCN) feedback to the sender will echo these counters.

TCP Accurate ECN Option:

Kind: TBD (same as above)

Length: 12 bytes

Kind	12	ECT(0)	ECT(1)	CE	non-ECT	loss	CE in bytes
1	1	2	2	1	1	1	3

Figure 2: Accurate ECN feedback TCP option

TCP anyway provides a mechanism to detect loss as loss should always be assumed as a strong signal for congestion and TCP congestion control reacts on loss. If TCP SACK is not available, the exact number of losses is not known. Moreover, the TCP loss detection (incl. SACK) is done in bytes and not in number of packets. The number of lost packets can be used by the sender to calculate the ECN Nonce sum more exactly.

The same feedback information are proposed for the (ECN) feedback in RTP (see [I-D.ietf-avtcore-ecn-for-rtsp]).

As TCP is a bi-directional protocol, this option can be used in both directions. With the reception of every data segment at least one of the counters changes (ECT(0) or ECT(1)). The AcceCN option SHOULD be included in every ACK to ensure the reception of the ECN feedback at the sender in case of ACK loss. To reduce network load the AcceCN option MAY not be sent in every ACK, e.g. only in very second ACK (if ACKs are sent very frequently).

In general it is possible that any of the counters wraps around. In this case the information might get corrupted if e.g. for any reason only one ACK per RTT is sent and more than 256 CE marks occur in one RTT. For this case it MUST be ensured, that at least three ACKs/segments with the AcceCN option have been sent prior to the counter experiencing an wrap around. Whenever an AcceCN Option is received with smaller counter value than in the previous one and the

respective ACK acknowledges new data, a wrap around MUST be assumed.

#### 4. Acknowledgements

#### 5. IANA Considerations

TBD

#### 6. Security Considerations

TBD

#### 7. References

##### 7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.

##### 7.2. Informative References

- [Ali10] Alizadeh, M., Greenberg, A., Maltz, D., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and M. Sridharan, "DCTCP: Efficient Packet Transport for the Commoditized Data Center", Jan 2010.
- [I-D.briscoe-tsvwg-re-ecn-tcp] Briscoe, B., Jacquet, A., Moncaster, T., and A. Smith, "Re-ECN: Adding Accountability for Causing Congestion to TCP/IP", draft-briscoe-tsvwg-re-ecn-tcp-09 (work in progress), October 2010.
- [I-D.ietf-avtcore-ecn-for-rtp] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", draft-ietf-avtcore-ecn-for-rtp-08 (work

in progress), May 2012.

[I-D.kuehlewind-conex-accurate-ecn]

Kuehlewind, M. and R. Scheffenegger, "Accurate ECN  
Feedback in TCP", draft-kuehlewind-conex-accurate-ecn-01  
(work in progress), October 2011.

#### Authors' Addresses

Mirja Kuehlewind (editor)  
University of Stuttgart  
Pfaffenwaldring 47  
Stuttgart 70569  
Germany

Email: [mirja.kuehlewind@ikr.uni-stuttgart.de](mailto:mirja.kuehlewind@ikr.uni-stuttgart.de)

Richard Scheffenegger  
NetApp, Inc.  
Am Euro Platz 2  
Vienna, 1120  
Austria

Phone: +43 1 3676811 3146  
Email: [rs@netapp.com](mailto:rs@netapp.com)



TCP Maintenance Working Group  
Internet-Draft  
Intended status: Experimental  
Expires: January 16, 2013

M. Mathis  
Google, Inc  
July 15, 2012

Laminar TCP and the case for refactoring TCP congestion control  
draft-mathis-tcpm-tcp-laminar-01.txt

## Abstract

The primary state variables used by all TCP congestion control algorithms, `cwnd` and `ssthresh`, are heavily overloaded, carrying different semantics in different states. This leads to excess implementation complexity and poorly defined behaviors under some combinations of events, such as application stalls during loss recovery. We propose a new framework for TCP congestion control, and to recast current standard algorithms to use new state variables. This new framework will not generally change the behavior of any of the primary congestion control algorithms when they are invoked in isolation. It will permit new algorithms with better behaviors in many corner cases, such as when two distinct primary algorithms are invoked concurrently. It will also foster the creation of new algorithms to address some events that are poorly treated by today's standards. For the vast majority of traditional algorithms the transformation to the new state variables is completely straightforward. However, the resulting implementation is likely to technically be in violation of existing TCP standards, even if it is fully compliant with their principles and intent.

## Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 16, 2013.

## Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Overview of the new algorithm . . . . .	3
3. Standards Impact . . . . .	4
4. Meta Language . . . . .	5
5. State variables and definitions . . . . .	6
6. Updated Algorithms . . . . .	6
6.1. Congestion avoidance . . . . .	7
6.2. Proportional Rate Reduction . . . . .	8
6.3. Restart after idle, Congestion Window Validation and Pacing . . . . .	8
6.4. RTO and F-RTO . . . . .	9
6.5. Undo . . . . .	10
6.6. Control Block Interdependence . . . . .	10
6.7. New Reno . . . . .	10
7. Example Pseudocode . . . . .	11
8. Compatibility with existing implementations . . . . .	12
9. Security Considerations . . . . .	13
10. IANA Considerations . . . . .	14
11. References . . . . .	14
Author's Address . . . . .	15

## 1. Introduction

The primary state variables used by all TCP congestion control algorithms, `cwnd` and `ssthresh`, are heavily overloaded, carrying different semantics in different states. Multiple algorithms sharing the same state variables lead to excess complexity, conflicting correctness constraints, and makes it unreasonably difficult to implement, test and evaluate new algorithms.

We are proposing a new framework for TCP congestion control that separate transmission scheduling, which determines precisely when data is sent, from pure congestion control, which determines the amount of data to be sent in each RTT. This separation is implemented with new state variables and greatly simplifies the interactions between the two subsystems. It permits vast range of new algorithms that are not feasible with the current parameterization.

This note describes the new framework and presents a preliminary mapping between current standards and new algorithms based on the new state variables. At this point the new algorithms are not fully specified, and many have still unconstrained design choices. In most cases, our goal is to precisely mimic today's standard TCP, at least as far as well defined primary behaviors. In general, it is a non-goal to mimic behaviors in poorly defined corner cases, or other cases where standard behaviors are viewed as being problematic.

It is called Laminar because one of its design goals is to eliminate unnecessary turbulence introduced by TCP itself.

## 2. Overview of the new algorithm

The new framework separates transmission scheduling, which determines precisely when data is sent, from pure Congestion Control, which determines the total amount of data sent in any given RTT.

The default algorithm for transmission scheduling is a strict implementation of Van Jacobsons' packet conservation principle [Jacobson88]. Data arriving at the receiver cause ACKs which in turn cause the sender to transmit an equivalent quantity of data back into the network. The primary state variable is implicit in the quantity of data and ACKs circulating in the network. This state observed through an improved "total\_pipe" estimator, which is based on "pipe" as described in RFC 3517 [RFC3517] but also includes the quantity of data reported by the current ACK and pending transmissions that have passed congestion control but are waiting for other events such as TSO.

A new state variable, CCwin, is the primary congestion control state variable. It is updated only by the congestion control algorithms, which are concerned with detecting and regulating the overall level of congestion along the path. CCwin is TCP's best estimate for an appropriate average window size. In general, it rises when the network seem to be underfilled and is reduced in the presence of congestion signals, such as loss, ECN marks or increased delay. Although CCwin resembles cwnd, cwnd is overloaded and used by multiple algorithms (such as burst suppression) with different and sometimes conflicting goals.

Any time total\_pipe is different from CCwin the transmission scheduling algorithm slightly adjusts the number of segments sent in response to each ACK. Slow start and Proportional Rate Reduction [PRRid] are both embedded in the transmission scheduling algorithm.

If CCwin is larger than total\_pipe, the default algorithm to grow total\_pipe is for each ACK to trigger one segment of additional data. This is essentially an implicit slowstart, but it is gated by the difference between CCwin and total\_pipe, rather than the difference between cwnd and ssthresh. In the future, additional algorithms such as pacing, might be used to raise total\_pipe.

During Fast Retransmit, the congestion control algorithm, such as CUBIC, generally reduces CCwin in a single step. Proportional Rate Reduction [PRRid] is used to gradually reduce total\_pipe to agree with CCwin. PRR was based on Laminar principles, so its specification has many parallels to this document.

Connection startup is accomplished as follows: CCwin is set to MAX\_WIN (akin to ssthresh), and IW segments are transmitted. The ACKs from these segments trigger additional data transmissions, and slowstart proceeds as it does today. The very first congestion event is a special case because there is not a prior value for CCwin. By default and on the first congestion event only, CCwin would be set from total\_pipe, and then standard congestion control is invoked.

The primary advantage of the Laminar framework is that by partitioning congestion control and transmission scheduling into separate subsystems, each is subject to simpler design constraints, making it far easier to develop many new algorithms that are not feasible with the current organization of the code.

### 3. Standards Impact

Since we are proposing to refactor existing standards into new state variables, all of the current congestion control standards documents

will potentially need to be reviewed. Although there are roughly 60 RFCs that mention `cwnd` or `ssthresh`, most only need self evident reinterpretation. Others, such as MIBs, warrant a sentence or two clarifying how to map `CCwin` and `total_pipe` onto existing specifications that use `cwnd` and `ssthresh`. There are however several RFCs that explicitly address the interplay between `cwnd` and `ssthresh` in today's TCP, including RFC 5681 [RFC5681], RFC 5682 [RFC5682], RFC 4015 [RFC4015], and RFC 6582 [RFC6582]. These need to be reviewed more carefully. In most cases the algorithms can easily be restated under the Laminar framework. Others, such as Congestion Window Validation [RFC2861], potentially require redesign.

This document does not propose to change the TCP friendly paradigm [RFC2914]. By default all updated algorithms using these new state variables would have behaviors similar to the current TCP implementations, however over the longer term the intent is to permit new algorithms that are not feasible today. For example, since `CCwin` does not directly affect transmissions during recovery, it is straightforward to permit recovery ACKs to raise `CCwin` even while PRR is reducing `total_pipe`. This facilitates so-called "fluid model" algorithms which further decouple congestion control from the details of the TCP the protocol.

But even without these advanced algorithms, we do anticipate some second order effects. For example while testing PRR it was observed that suppressing bursts by slightly delaying transmissions can improve average performance, even though in a strict sense the new algorithm is less aggressive than the old [IMC11PRR].

#### 4. Meta Language

We use the following terms when describing algorithms and their alternatives:

**Standard** - The current state of the art, including both formal standards and widely deployed algorithms that have come into standard use, even though they may not be formally specified. [Although PRR does not yet technically meet these criteria, we include it here].

**default** - The simplest or most straightforward algorithm that fits within the Laminar framework. For example implicit slowstart whenever `total_pipe` is less than `CCwin`. This term does not make a statement about the relative aggressiveness or any other properties of the algorithm except that it is a reasonable choice and straightforward to implement.

**conformant** - An algorithm that can produce the same packet trace as a

TCP implementation that strictly conforms to the current standards.

mimic - An algorithm constructed to be conformant to standards.

opportunity - An algorithm that can do something better than the standard algorithm, typically better behavior in a corner cases that is either not well specified or where the standard behavior is viewed as being less than ideal.

more/less aggressive - Any algorithm that sends segments earlier/later than another (typically conformant) algorithm under identical sequences of events. Note that this is an evaluation of the packet level behavior, and does not reflect any higher order effects.

Observed performance - A statement about algorithm performance based on a measurement study or other observations based on a significant sample of authentic Internet paths. e.g. an algorithm might have observed data rate that is different than another (typically conformant) algorithm.

application stall - The application is failing to keep up with TCP: either the sender is running out of data to send, or the receiver is not reading it fast enough. When there is an application stall, congestion control does not regulate data transmission and some of the protocol events are triggered by application reads or writes, as appropriate.

## 5. State variables and definitions

CCwin - The primary congestion control state variable.

DeliveredData - The total number of bytes that the current ACK indicates have been delivered to the receiver. (See [PRRid] for more details).

total\_pipe - The total quantity of circulating data and ACKs. In addition to RFC 3517 pipe, it includes DeliveredData for the current ack, plus any data held for delayed transmission, for example to permit a later TSO transmission.

sendcnt - The quantity of data to be sent in response to the current ACK or other event.

## 6. Updated Algorithms

A survey of standard, common and proposed algorithms, and how they

might be reimplemented under the Laminar framework.

#### 6.1. Congestion avoidance

Under the Laminar framework the loss recovery mechanism does not, by default, interfere with the primary congestion control algorithms. The CCwin state variable is updated only by the algorithms that decide how much data to send on successive round trips. For example standard Reno AIMD congestion control [RFC5681] can be implemented by raising CCwin by one segment every CCwin worth of ACKs (once per RTT) and halving it on every loss or ECN signal (e.g.  $CCwin = CCwin/2$ ). During recovery the transmission scheduling part of the Laminar framework makes the necessary adjustments to bring total\_pipe to agree with CCwin, without tampering with CCwin.

This separation between computing CCwin and transmission scheduling will enable new classes of congestion control algorithms, such as fluid models that adjust CCwin on every ACK, even during recovery. This is safe because raising CCwin does not directly trigger any transmissions, it just steers the transmission scheduling closer to the end of recovery. Fluid models have a number of advantages, such as simpler closed form mathematical representations, and are intrinsically more tolerant to reordering since non-recovery disordered states don't inhibit window growth.

Investigating alternative algorithms and their impact is out of scope for this document. It is important to note that while our goal here is not to alter the TCP friendly paradigm, Laminar does not include any implicit or explicit mechanism to prevent a Tragedy of the Commons. However, see the comments in Section 9.

The initial slowstart does not use CCwin, except that CCwin starts at the largest possible value. It is the transmission scheduling algorithms that are responsible for performing the slowstart. On the first loss it is necessary to compute a reasonable CCwin from total\_pipe. Ideally, we might save total\_pipe at the time each segment is scheduled for transmission, and use the saved value associated with the lost segment to prime CCwin. However, this approach requires extra state attached to every segment in the retransmit queue. A simpler approach is to have a mathematical model the slowstart, and to prime CCwin from total\_pipe at the time the loss is detected, but scaled down by the effective slowstart multiplier (e.g. 1.5 or 2). In either case, once CCwin is primed from total\_pipe, it is typically appropriate to invoke the reduction on loss function, to reduce it again per the congestion control algorithm.

Nearly all congestion control algorithms need to have some mechanism

to prevent CCwin from growing while it is not regulating transmissions e.g. during prolonged application stalls.

## 6.2. Proportional Rate Reduction

Since PRR [PRRid] was designed with Laminar principles in mind, updating it is a straightforward variable substitution. CCwin replaces ssthresh, and RecoverFS is initialized from total\_pipe at the beginning of recovery. Thus PRR provides a gradual window reduction from the prior total\_pipe down to the new CCwin.

There is one important difference from the current standards: CCwin is computed solely on the basis of the prior value of CCwin. Compare this to RFC 5681 which specifies that the congestion control function is computed on the basis of the FlightSize (e.g.  $ssthresh = \text{FlightSize}/2$  ). This change from prior standard completely alters how application stalls interact with congestion control.

Consider what happens if there is an application stall for most of the RTT just before a Fast Retransmit: Under Laminar it is likely that CCwin will be set to a value that is larger than total\_pipe, and subject to available application data PRR will go directly to slowstart mode, to raise total\_pipe up to CCwin. Note that the final CCwin value does not depend on the duration of the application stall.

With standard TCP, any application stall reduces the final value of cwnd at the end of recovery. In some sense application stalls during recovery are treated as though they are additional losses, and have a detrimental effect on the connection data rate that lasts far longer than the stall itself.

If there are no application stalls, the standard and Laminar variants of the PRR algorithm should have identical behaviors. Although it is tempting to characterize Laminar as being more aggressive than the standards, it would be more apropos to characterize the standard as being excessively timid under certain combinations of overlapping events that are not well represented by benchmarks or models.

## 6.3. Restart after idle, Congestion Window Validation and Pacing

Decoupling congestion control from transmission scheduling permits us to develop new algorithms to raise total\_pipe to CCwin after an application stall or other events. Although it was stated earlier that the default transmission scheduling algorithm for raising total\_pipe is an implicit slowstart, there is opportunity for better algorithms.

We imagine a class of hybrid transmission scheduling algorithms that

use a combination of pacing and slowstart to reestablish TCP's self clock. (See [Visweswaraiah99].) For example, whenever `total_pipe` is significantly below `CCwin`, `RTT` and `CCwin` can be used to directly compute a pacing rate. We suspect that pacing at the previous full rate will prove to be somewhat brittle, sometimes causing excessive loss and yielding erratic results. It is more likely that a hybrid strategy will work better and be better for the network, for example by pacing at some fraction ( $1/2$  or  $1/4$ ) of the prior rate until `total_pipe` reaches some fraction of `CCwin` (e.g.  $CCwin/2$ ) and then using conventional slowstart to bring `total_pipe` the rest of the way up to `CCwin`.

This is far less aggressive than standard TCP without `cwnd` validation [RFC2861] or when the application stall was less than one `RTO`, since standards permit TCP to send a full `cwnd` size burst in these situations. It is potentially more aggressive than conventional slowstart invoked by `cwnd` validation when the application stall is longer than several `RTOs`. Both standard behaviors in these situations have always been viewed as problematic, because interface rate bursts are clearly too aggressive and a full slowstart is clearly too conservative. Mimicking either is a non-goal, when there is ample opportunity to find a better compromise.

Although strictly speaking any new transmission scheduling algorithms are independent of the Laminar framework, they are expected to have substantially better behavior in many common environments and as such strongly motivate the effort required to refactor TCP implementations and standards.

#### 6.4. `RTO` and `F-RTO`

We are not proposing any changes to the `RTO` timer or the `F-RTO` [RFC5682] algorithm used to detect spurious retransmissions. Once it is determined that segments were lost, `CCwin` is updated to a new value as determined by the congestion control function, and Laminar implicit slowstart is used to clock out (re)transmissions. Once all holes are filled, a hybrid paced transmissions can be used to reestablish TCPs self clock at the new data rate. This can be the same hybrid pacing algorithm as is used to recover the self clock after application stalls.

Note that as long as there is non-contiguous data at the receiver the retransmission algorithms require timely SACK information to make proper decisions about which segments to send. Pacing during loss recovery is not recommended without further investigation.

### 6.5. Undo

Since CCwin is not used to implement transmission scheduling, undo is trivial. CCwin can just be set back to its prior value and the transmission scheduling algorithm will transmit more (or less) data as needed. It is useful to note that the discussion about ssthresh in [RFC4015] also applies to CCwin in TCP Laminar. Some people might find it useful to think of CCwin as being equivalent to `MAX(ssthresh,cwnd)`.

There is an opportunity to do substantially better than current algorithms. Undo can be implemented by saving the arithmetic difference between the current and prior value of CCwin, and then adding this delta back into CCwin when all retransmissions are deemed to be spurious. If the congestion avoidance algorithm is linear (or can be linearized), and is mathematically transportable across undo, it is possible to design a congestion control algorithm that is completely immune to reordering in the sense that the overall evolution of CCwin is not affected by low level reordering, even if it is pervasive. This is an area for future research.

### 6.6. Control Block Interdependence

Under the Laminar framework, congestion control state can be easily shared between connections [RFC2140]. An ensemble of connections can each maintain their own `total_pipe` (partial\_pipe?) which in aggregate tracks a single common CCwin. A master transmission scheduler allocates permission to send (`sndcnt`) to each of the constituent connection on the basis of the difference between the CCwin and the aggregate `total_pipe`, and a fairness or capacity allocation policy that balances the flows. Note that ACKs on one connection in an ensemble might be used to clock transmissions on another connection, and that following a loss, the window reductions can be allocated to flows other than the one experiencing the loss.

### 6.7. New Reno

The key to making Laminar function well without SACK is having good estimators for `DeliveredData` and `total_pipe`. By definition every duplicate ACK indicates that one segment has arrived at the receiver and `total_pipe` has fallen by one. On any ACK that advances `snd.una`, `total pipe` can be updated from `snd.nxt-snd.una`, and `DeliveredData` is the change in `snd.una`, minus the sum of the estimated `DeliveredData` of the preceding duplicate ACKs. As with SACK the total `DeliveredData` must agree with the overall forward progress over time.

## 7. Example Pseudocode

On startup:

```
CCwin = MAX_WIN  
sndBank = IW
```

On every ACK:

```
DeliveredData = delta(snd.una) + delta(SACKd)  
pipe = (RFC 3517 pipe algorithm)  
total_pipe = pipe+DeliveredData+sndBank  
sndcnt = DeliveredData    // Default # transmissions  
  
if new_recovery():  
    if CCwin == MAX_WIN:  
        CCwin = total_pipe/2 // First time only  
        CCwin = CCwin/2      // Reno congestion control  
        prr_delivered = 0    // Total bytes delivered during recov  
        prr_out = 0         // Total bytes sent during recovery  
        RecoverFS = total_pipe //  
  
if !in_recovery() && !application_limited():  
    CCwin += (MSS/CCwin)  
    prr_delivered += DeliveredData // noop if not in recovery
```

```
if total_pipe > CCwin:
    // Proportional Rate Reduction
    sndcnt = CEIL(prr_delivered * CCwin / RecoverFS) - prr_out

else if total_pipe < CCwin:
    if in_recovery():
        // PRR Slow Start Reduction Bound
        limit = MAX(prr_delivered - prr_out, DeliveredData) + SMSS
        sndcnt = MIN(CCwin - total_pipe, limit)
    else:
        // slow start with appropriate byte counting
        inc = MIN(DeliveredData, 2*MSS)
        sndcnt = DeliveredData + inc

// cue the transmission machinery
sndBank += sndcnt
limit = maxBank()
if sndBank > limit:
    sndBank = limit
tcp_output()
```

For any data transmission or retransmission:

```
tcp_output():
    while sndBank && tso_ok():
        len = sendsomething()
        sndBank -= len
        prr_out += len // noop if not in recovery
```

## 8. Compatibility with existing implementations

On a segment by segment basis, the above algorithm is [believed to be] fully conformant with or less aggressive than standards under all conditions.

However this condition is not sufficient to guarantee that observed performance can't be better than standards. Consider an application that keeps TCP in bulk mode nearly all of the time, but has occasional pauses that last some fraction of one RTT. A fully conformant TCP would be permitted to "catch up" by sending a partial window burst at full interface rate. On some networks, such bursts might be very disruptive, causing otherwise unnecessary packet losses and corresponding cwnd reductions.

In Laminar the default algorithm would be slowstart. Other algorithms that might cause the same bursts would be permitted, although are not described here. A better algorithm would be to pace the data at (some fraction of) the prior rate. Neither pacing nor slowstart is likely to cause unnecessary losses, and as was observed while testing PRR, being less aggressive at the segment level has the potential to increase the observed performance[IMC11PRR]. In this scenario Laminar with pacing has the potential to outperform both of the behaviors described by standards.

## 9. Security Considerations

The Laminar framework does not change the risk profile for TCP (or other transport protocols) themselves.

However, the complexity of current algorithms as embodied in today's code present a substantial barrier to people wishing to cheat "TCP friendliness". It is a fairly well known and easily rediscovered result that custom tweaks to make TCP more aggressive in one environment generally make it fragile and perform less well across the extreme diversity of the Internet. This negative outcome is a substantial intrinsic barrier to wide deployment of rogue congestion control algorithms.

A direct consequence of the changes proposed in this note, decoupling congestion control from other algorithms, is likely to reduce the barrier to rogue algorithms. However this separation and the ability to introduce new congestion control algorithms is a key part of the motivation for this work.

It is also important to note that web browsers have already largely defeated TCP's ability to regulate congestion by opening many concurrent connections. When a Web page contains content served from multiple domains (the norm these days) all modern browsers open between 35 and 60 connections (see: <http://www.browserscope.org/?category=network> ). This is the Web community's deliberate workaround for TCP's perceived poor performance and inability make full use of certain types of consumer grade networks. As a consequence the transport layer has already lost a substantial portion of its ability to regulate congestion. It was not anticipated that the tragedy of the commons in Internet congestion would be driven by competition between applications and not between TCP implementations.

In the short term, we can continue to try to use standards and peer pressure to moderate the rise in overall congestion levels, however the only real solution is to develop mechanisms in the Internet

itself to apply some sort of backpressure to overly aggressive applications and transport protocols. We need to redouble efforts by the ConEx WG and others to develop mechanisms to inform policy with information about congestion and its causes. Otherwise we have a looming tragedy of the commons, in which TCP has only a minor role.

Implementers that change Laminar from counting bytes to segments have to be cautious about the effects of ACK splitting attacks[Savage99], where the receiver acknowledges partial segments for the purpose of confusing the sender's congestion accounting.

## 10. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

## 11. References

- [Jacobson88] Jacobson, V., "Congestion Avoidance and Control", SIGCOMM 18(4), August 1988.
- [RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140, April 1997.
- [RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window Validation", RFC 2861, June 2000.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.
- [RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC 3517, April 2003.
- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC5682] Sarolahti, P., Kojo, M., Yamamoto, K., and M. Hata, "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP", RFC 5682,

September 2009.

- [RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, April 2012.
- [PRRid] Mathis, M., Dukkupati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", draft-mathis-tcpm-proportional-rate-reduction-01 (work in progress), July 2011.
- [IMC11PRR] Mathis, M., Dukkupati, N., Cheng, Y., and M. Ghobadi, "Proportional Rate Reduction for TCP", Proceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference , 2011.
- [Savage99] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP congestion control with a misbehaving receiver", SIGCOMM Comput. Commun. Rev. 29(5), October 1999.
- [Visweswaraiah99] Visweswaraiah, V., "Improving Restart of Idle TCP Connections", Tech Report USC TR 97-661, November 1997.

#### Author's Address

Matt Mathis  
Google, Inc  
1600 Amphitheater Parkway  
Mountain View, California 93117  
USA

Email: mattmathis@google.com



Network Working Group  
Internet-Draft  
Intended Status: Standards Track  
Expires: February 14, 2013

A. Sabatini  
Broker Communications Inc.  
.  
August 15, 2012

Highly Efficient Selective Acknowledgement (SACK) for TCP  
draft-sabatini-tcp-sack-01

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at  
<http://www.ietf.org/lid-abstracts.html>

The list of Internet-Draft Shadow Directories can be accessed at  
<http://www.ietf.org/shadow.html>

This Internet-Draft will expire on January 22, 2013.

Comments are solicited and should be addressed to the author at  
[draft-sack@tsabatini.com](mailto:draft-sack@tsabatini.com).

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Abstract

This memo expands on the Selective Acknowledgement Protocol described in RFC2018 to improve its performance and efficiency while reducing the delay involved in recovering lost segments. This leads to very reliable and efficient communications regardless of transit delay or high levels of lost segments due to noise or congestion. It introduces a fundamentally new way of looking at Selective Acknowledgement and uses this concept to improve the performance of the RFC2018 protocol. This memo proposes an implementation of the improved SACK and discusses its performance and related issues.

## Acknowledgements

Much of the text in this document is taken directly from RFC2018 "TCP Selective Acknowledgement Options" by M. Mathis, J. Mahdavi, S. Floyd and A. Romanow and RFC1072 "TCP Extensions for Long-Delay Paths" by B. Braden and V. Jacobson.

## 1. Introduction

This revision to the SACK protocol has its roots in a similar, HDLC based protocol I designed and implemented for secure financial transactions. That protocol, being designed for use on a worldwide basis, was born out of the need for a protocol that would handle any communications environment no matter how noisy or how much delay (including multiple satellite hops) was in the path. In later years its properties were found valuable in congestion situations where packets were dropped.

Multiple packet losses from a window of data can have a catastrophic effect on TCP throughput. TCP [Postel81] uses a cumulative acknowledgment scheme in which received segments that are not at the left edge of the receive window are not acknowledged. This forces the sender to either wait a round-trip time to find out about each lost packet, or to unnecessarily retransmit segments which have been correctly received [Fall95]. With the cumulative acknowledgment scheme, multiple dropped segments generally cause TCP to lose its ACK-based clock, reducing overall throughput.

Selective Acknowledgment (SACK) is a strategy which corrects this behavior in the face of multiple dropped segments. With selective acknowledgments, the data receiver can inform the sender about all segments that have arrived successfully, so the sender need retransmit only the segments that have actually been lost. The compatible extensions to RFC2018 proposed here enhance the protocol by changing retransmission from a worst case timer basis to a deterministic, state driven basis which responds rapidly to link conditions.

I propose modifications to the SACK options as proposed in RFC2018. Specifically, I add a transmit state to each transmitted message and return that transmit state when each acknowledgement is sent. By using the returned transmit state I can tell what messages have been transmitted after the information in the acknowledgement and thus rebuild the current state of the receiver at the transmitter. I also propose changes to the way SACK blocks are reported to insure that the oldest, and thus the most critical, are transmitted expeditiously without jeopardizing the multiple repetition of SACK information which gives the current protocol its reliability. Additionally since the space to store acknowledgements in IPv4 is limited and may not be able to accommodate all of the acknowledgement pairs, I propose a method of sending the complete receiver state by sending multiple acknowledgements when it becomes evident that transmission has stalled due to loss of multiple ACKs.

The RFC2018 selective acknowledgment extension uses two TCP options. The first is an enabling option, "SACK-permitted", which may be sent in a SYN segment to indicate that the SACK option can be used once the connection is established. This option is extended to both indicate that this newer version of the protocol is being used and to establish an initial value for transmit state. The other is the SACK option itself, which may be sent over an established connection once permission has been given by SACK-permitted. This has also been extended to add both the transmit state implicit in the message and the transmit state that was received at the far end (now called "Returned State").

The SACK option is to be included in a segment sent from a TCP that is receiving data to the TCP that is sending that data; we will refer to these TCP's as the data receiver and the data sender, respectively. We will consider a particular simplex data flow; any data flowing in the reverse direction over the same connection can be treated independently.

## 2. Underlying concepts

In order for a sender to know how to optimally transmit messages to a receiver the sender must recreate the state of the receiver as of the last acknowledgement received (which segments have been received and acknowledged, which segments have not) and then "age" or modify that state by updating it based upon the messages transmitted since the state implicit in the acknowledgement was current. In order to do this the sender must maintain a transmission order list which contains entries for the segment ranges of each message as it is sent. We called the index into the transmission order list "Send State" and transmit this state variable with each message. The receiver, after correctly receiving the message, saves this value and

returns it (now called "Returned State") and the list of selectively acknowledged segments with each acknowledgement. When the sender receives this information it is then capable of constructing a list of missing segments by taking its unacknowledged segment range list and modifying it on the basis of the received selective acknowledgements and then removing from that list all segments that have been transmitted since the message which caused the acknowledgement which is all segments sent with indexes between the current "send state" and the "receive state" in the acknowledgement message.

To accommodate the issue of receiving segments out of order at the receiver, or those packets delayed by alternate routing, the sender does not instantly update its Current Returned State value from the incoming ACK (which could trigger a false retransmission) but rather puts it on a timer queue for a length of time ("Reordering Time") appropriate to the delay randomness in the arrival path (typically 20 to 100 ms based on media, speed and distance), which when the timer entry expires, causes the update of the Current Returned State value. If the updated Current Returned State value shows blocks that remain unacknowledged after this time out they are assumed to be lost and they are queued for retransmission.

Thus by transmitting the complete acknowledgement information through the SACK blocks from the receiver along with an indicator to the sender as to its state current at the time of the acknowledgement the sender can accurately recreate the current status of the receiver assuming all "in flight" messages were received and thus only send the unacknowledged messages starting with the oldest followed by any new messages whose transmission is requested.

### 3. Enhanced Sack-Permitted Option

This document is designed to be an extension of RFC2018 and any implementation of it must be designed to fall back to handling RFC2018 when the other party is not capable of handling the enhanced protocol.

Although Enhanced SACK is a compatible extension of standard SACK it is recognized that certain middleware boxes are not RFC compliant as to extensions and therefore will fail if, as would properly be done, Enhanced SACK was handled as such. Therefore Enhanced SACK is transmitted as two options both in the SYN packets as well as data and ACK packets.

The first option of the pair MUST be the standard SACK option (Kind = 4) if this endpoint desires a SACK session of any kind. The second four or six byte option may be sent in a SYN by a TCP that has been

extended to receive (and presumably process) the Enhanced SACK option in order to indicate its willingness to enter into an Enhanced SACK session.

This option MUST NOT be transmitted on non-SYN segments in the current protocol, it is left to future study as to its use for transmitting long sequences of acknowledgements in one frame.

TCP SACK-Permitted Option:

Kind: 4

0										1										2										3									
0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9
										Kind=4										Length=2																			

TCP Enhanced SACK Setup Option:

Kind: X1

Kind=X1										Length=6 or 8										P	R	P	R	Reserved									
																				T	T	X	X										
																				O	O	T	T										

if RXT = 0 not requesting extended state

Send State Token																			
------------------	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

if RXT = 1 requesting extended state

Extended Send State Token										Reserved									
---------------------------	--	--	--	--	--	--	--	--	--	----------	--	--	--	--	--	--	--	--	--

Control Bits: 4 bits (from left to right):

PTO: Tokens Permitted, the sender supports the requirements of this extension

RTO: Request Tokens, sender requests link use the protocol outlined in this extension

PXT: Extended Tokens Supported

RXT: Request Extended Tokens: sender requests using extended

tokens.

Reserved: 12 bits

Reserved for future use, must be set to zero

If RTO is set then the Send State immediately follows, 16 bits if RXT is not set and 24 bits if it is. If necessary the option is padded with a binary zero byte so that length is an even number. In the case that one end can only support 16 bit tokens only the right most 16 bits of the extended field is used.

For brevity in this document only the lesser, 16 bit format is shown.

#### 4. SACK Option Format

As with the SYN options, Enhanced SACK information must be transmitted as a separate option in order to accomodate non RFC compliant middleware boxes. By its nature it must precede the TCP SACK Option.

TCP Enhanced SACK Option:

Kind: X2

Length: 6 (or 8 if extended token)

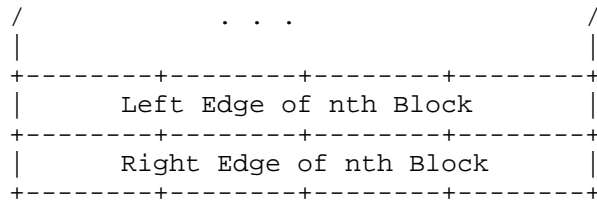
Kind=X2		Len=6	
Send State	Returned State		

TCP SACK Option:

Kind: 5

Length: Variable

Kind=5	Length
Left Edge of 1st Block	
Right Edge of 1st Block	



The SACK option is to be sent by a data receiver to inform the data sender of non-contiguous blocks of data that have been received and queued. The data receiver awaits the receipt of data (perhaps by means of retransmissions) to fill the gaps in sequence space between received blocks. When missing segments are received, the data receiver acknowledges the data normally by advancing the left window edge in the Acknowledgement Number Field of the TCP header. The SACK option does not change the meaning or use of the Acknowledgement Number field.

This option contains a list of some of the blocks of contiguous sequence space occupied by data that has been received and queued within the window.

Each contiguous block of data queued at the data receiver is defined in the SACK option by two 32-bit unsigned integers in network byte order:

- \* Left Edge of Block

This is the first sequence number of this block.

- \* Right Edge of Block

This is the sequence number immediately following the last sequence number of this block.

Each block represents received bytes of data that are contiguous and isolated; that is, the bytes just below the block, (Left Edge of Block - 1), and just above the block, (Right Edge of Block), have not been received.

A SACK option that specifies  $n$  blocks will have a length of  $8n+6$  bytes, so the 40 bytes available for TCP options can specify a maximum of 4 blocks. It is suggested that the Enhanced SACK will provide the time-stamp information used for RTTM [Jacobson92].

## 5. Generating Sack Options: Data Receiver Behavior

If the data receiver has received a SACK-Permitted option on the SYN

for this connection, the data receiver MAY elect to generate SACK options as described below. If the data receiver generates SACK options under any circumstance, it MUST generate them under all permitted circumstances. If the data receiver has not received a SACK-Permitted option for a given connection, it MUST NOT send SACK options on that connection.

If sent at all, SACK options MUST be included in all ACKs which do not ACK the highest sequence number in the data receiver's queue. In this situation the network has lost or mis-ordered data, such that the receiver holds non-contiguous data in its queue. RFC 1122, Section 4.2.2.21, discusses the reasons for the receiver to send ACKs in response to additional segments received in this state. The receiver MUST send an ACK for every valid segment that arrives containing new data, and each of these "duplicate" ACKs SHOULD bear a SACK option.

The purpose of the SACK blocks is to recreate the status of the receiver at the transmitter. To that end the most important information is (1) new or changed blocks, (2) the second transmission of new or changed blocks, (3) a complete enumeration of all received blocks starting from the oldest first.

If the data receiver chooses to send a SACK option, the following rules apply:

- \* The data receiver first fills in "Send State" in the option from the current value of its "Send State". The data receiver then fills in "Returned State" from its "Saved Send State" which was set by either the SYN option or the SACK option of the last TCP packet that contained a value which was logically greater than the current saved value.

- \* SACK blocks representing all discontinuous segment ranges received where those ranges are logically over the Acknowledgement Number in the TCP header are kept in logically ascending by segment range list. Additionally a count (a four byte binary for safety) is maintained in each block which represents the number of times it has been transmitted. Each time a SACK block is added or changes (normally by being merged with another entry) the count is set to zero(0).

- \* All SACK block slots SHOULD be filled on each each normal ACK transmitted, starting with those that have the lowest count (acknowledging the most recently received segments), followed by those with the next lowest count, and so on until all SACK block slots are filled. As each SACK block is moved to a slot its count is incremented by one(1) (thus care needs to be taken on the

second and subsequent passes to skip those entries currently in SACK blocks slots). By always starting from the oldest we insure the most critical have the first chance at receiving a SACK block slot. SACK blocks are empty ONLY if there are less SACK blocks outstanding than there are available slots. This methodology assures a fair number of transmissions to all SACK blocks.

\* The receiver receives a Send State from the sender that is logically greater than any previously seen the receiver must generate an ACK regardless of whether any SACK blocks have changed. Note that such a Send State change can come from an ACK produced by the sender as well as a message.

\* The definitions in RFC1022 are changed such that if there are entries on the SACK block list an ACK ALWAYS goes out in response to a received data segment.

\* To insure that the last added or changed SACK block is transmitted a second time, if the link goes idle for 2\*Reordering Time the receiver SHOULD send another ACK following the rules above.

\* A timer is maintained based on the timestamp of the oldest SACK block and is set to  $RTT * 1.25$ , it is reset each time a SACK block with a different segment start becomes the oldest SACK block. At the expiration of this timer, since this and probably other segments have not been retransmitted to the receiver, the receiver resets the timer to  $.25 * RTT$  and again sends sufficient acknowledgements to completely transmit all current SACK blocks starting from the one with the logically lowest segment start and proceeding in ascending sequence. Note that this process is aborted by any action that changes the oldest SACK block. This timer is used to assure that in case of a burst error the sender has enough information to restart properly.

#### 6. Interpreting the Sack Option and Retransmission Strategy: Data Sender Behavior

As each transmission request from the calling program is processed and entry is made into the segment queue to handle the request and its buffering. Entries are removed from the segment queue when the segment is completely acknowledged either through the Acknowledgement Number Field of the TCP header passing through the end of that segment or by a SACK block completing the acknowledgement of that segment. The segment is also appended to the end of the retransmission queue and transmission restarted from that segment if transmission has stopped.

Before processing the SACK information the Acknowledgement Number Field of the TCP header is used to eliminate outdated entries from the segment queue, saved list and retransmission queue before new information is added. The acknowledgement may split the first entry in either the segment queue or the retransmission queue in which case a pseudo entry is created in that queue for the unacknowledged remainder which additionally points to the saved original entry with an additional field which is the count of pseudo segments derived from it, set to one in this case. The Acknowledgement Number Field of the TCP header may end up eliminating a pseudo entry in which case the pseudo segment count of the original saved entry is decremented and if zero the segment is then entirely removed from both the segment queue and the saved list.

In processing selective acknowledgements the transmitter applies each SACK block to first the segment queue and then the retransmission queue. If the SACK block completely acknowledges a segment it is removed from the segment queue and moved to the saved list with a count of zero or completely if from the retransmission queue. If the SACK block completely acknowledges a pseudo segment that segment is removed and if from the segment queue the pseudo segment count in the related saved entry is decremented. If the SACK block acknowledges the beginning or end of a segment in either queue a pseudo entry is created with the adjusted unacknowledged remainder, if the segment was on the segment list the original segment is moved to the saved list and the pseudo count is set to one. If the entry was already a pseudo segment and this SACK acknowledges the beginning or end of the segment, the segment limits are adjusted but no other action occurs. If this entry is already a pseudo entry and the SACK block splits the segment in two a second pseudo entry is created to handle the right hand side of the range and, if on the segment list, the pseudo segment count in the related save list entry is incremented by one. The original pseudo entry is modified to represent the left hand range created by the SACK.

The sender maintains a Transmission List which an array of structures into which the segment start and end addresses of each transmitted block (be it a primary transmission or a retransmission) is placed. This list is, for optimal processing, a power of 2 in size and is, at a minimum, four times as large as the Maximum Number of Segments Outstanding. As each segment is transmitted the current Transmit Token modulus the Transmission List size is used as an index into this structure to store the segment start and end and then the Transmit Token is incremented by one.

When each each new SACK option is processed, its Returned Token is checked against the Current Returned Token and the Future Returned Token List, if logically greater than any of the above, it is

inserted into the Future Returned Token list in logical order along with the current time-stamp incremented by the Reordering Time. If it is the first entry on the list a timer is started for the value token time stamp minus current time stamp. When the timer expires the first entry on the Future Returned Token list set as the Current Returned Token and then it is removed from the list. If there are more members on the Future Returned Token list the timer is restarted with a value of the time-stamp in that entry minus the current time-stamp. A change in the Current Returned Token causes a recreation of the Retransmission Queue by first copying the Segment Queue and then removing from it all segments that have been transmitted subsequently, in a process identical to processing a SACK block starting at the segment identified by the Current Returned Token from the Transmission List and continuing through the Transmission List up to the (but not including) the Transmit Token. The Transmission Pointer is then set to first incomplete entry in the Retransmission Queue and transmission restarted if it has stopped.

Another method of implementation which allows quicker retransmission response at the expense of building the Retransmission Queue a second time is to retrieve the time stamp of the just received Returned Token if that Returned Token has not previously been seen (by comparing it with the Current Returned Token and the Future Returned Token list) and then subtracting from that timestamp the Reordering Time to allow for out of order messages. This value is then used to select any entry on the Transmission List with an equal or greater timestamp. The first version of the Retransmission Queue is created by copying the Segment Queue and then removing the segments that have since been retransmitted based on the adjusted timestamp. When the timer on the Future Returned Token List expires the retransmission queue is recreated a second time as in the preceding paragraph.

Note: For processing efficiency we believe most people will implement the Retransmission Queue as additional fields in the Segment Queue.

## 6.1 Handling last segment problem

If the sender side of the link goes idle for  $2 \times \text{Reordering Time}$  and there are still unacknowledged segments the sender SHOULD send an ACK (which would have the updated Send State) so that the receiver may ultimately detect if the last message is missing and cause it to be transmitted (the receiver would pass the Send State back as Returned State and the sender would realize the segment is still outstanding).

## 6.2 Congestion Control Issues

This document does not attempt to specify in detail the congestion control algorithms for implementations of TCP with SACK. However,

the congestion control algorithms present in the de facto standard TCP implementations MUST be preserved [Stevens94]. This algorithm eliminates much unnecessary retransmission so is likely to lessen overall congestion.

Note that the enhanced protocol does not suffer from traditional congestion collapse even though it is more robust, since it does not use timers and is rate limited by the tokens. Delayed and lost messages and ACKS make it slower but do not increase the traffic it sends significantly.

The use of time-outs as a fall-back mechanism for detecting dropped packets is unchanged by the SACK option. Because in normal operation acknowledgements will prevent retransmit timeout, when a retransmit timeout occurs the data sender SHOULD ignore prior SACK information in determining which data to retransmit.

Future research into congestion control algorithms may take advantage of the additional information provided by SACK. One such area for future research concerns modifications to TCP for a wireless or satellite environment where packet loss is not necessarily an indication of congestion.

## 7. Efficiency and Worst Case Behavior

Although this high efficiency improved SACK option sends more and larger SACK blocks and more acknowledgements than the previous version, with an active bi-directional link additional acknowledgements are often associated with data transmission and thus not a penalty. If the SACK option needs to be used due to segment loss then the improved efficiency afforded with this protocol more than justifies the additional SACK blocks.

The deployment of other TCP options may reduce the number of available SACK blocks to 2 or even to 1. This will reduce the redundancy of SACK delivery in the presence of lost ACKs. Even so, the exposure of TCP SACK in regard to the unnecessary retransmission of packets is strictly less than the exposure of current implementations of TCP. The worst-case conditions necessary for the sender to needlessly retransmit data is discussed in more detail in a separate document [Floyd96].

Older TCP implementations which do not have the SACK option will not be unfairly disadvantaged when competing against SACK-capable TCPs. This issue is discussed in more detail in [Floyd96].

## 8. Time-stamping

One pleasant benefit of having a token which is returned by the far end on a deterministic basis is the easy calculation of round trip delay. We can save a time stamp along with the segment information in our transmission order array. This allows us to calculate round trip delay when we receive our "Returned State" value and use it to access the time-stamp. Since more than one received message might have the same "Returned State" value we mark the time-stamp after use to indicate that the value should not be used again. Note that if an acknowledgement is lost we will calculate a longer delay than is accurate therefore we must smooth the returned values, typically returning the smallest out of the last N where N is typically four.

#### 9. Data Receiver Reneging

Since the Sender is recreating the state of the Receiver, the data Receiver MUST NOT discard data in its queue once that data has been reported in a SACK option. The Receiver is responsible for allocating enough buffers so that the missing segments within the window may be properly received and processed. Since enhanced SACK is event driven the lack of a new event for  $2.50 \times \text{RTT}$  SHOULD trigger a connection reset to guard against denial of service attacks.

#### 10. Security Considerations

This document neither strengthens nor weakens TCP's current security properties.

#### 11. References

[Jacobson88], Jacobson, V. and R. Braden, "TCP Extensions for Long-Delay Paths", RFC 1072, October 1988.

[Jacobson92] Jacobson, V., Braden, R., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[Mathis96] Mathis, M., Mahdavi, J., Floyd, S., Romanow, J. "TCP Selective Acknowledgement Options", RFC 2018, October 1996.

[Postel81] Postel, J., "Transmission Control Protocol - DARPA Internet Program Protocol Specification", RFC 793, DARPA, September 1981.

#### Author's Address

Anthony Sabatini  
Broker Communications Inc.  
200 West 20th Street

Suite 1216  
New York, NY 10011  
Email: draft-sack@tsabatini.com

The author is currently a master's degree candidate at -

Hofstra University  
Hempstead, N.Y.

His adviser is Dr. Xiang Fu

TCP Maintenance and Minor Extensions  
(tcpm)  
Internet-Draft  
Updates: 1323 (if approved)  
Intended status: Experimental  
Expires: April 25, 2013

R. Scheffenegger  
NetApp, Inc.  
M. Kuehlewind  
University of Stuttgart  
B. Trammell  
ETH Zurich  
October 22, 2012

Additional negotiation in the TCP Timestamp Option field  
during the TCP handshake  
draft-scheffenegger-tcpm-timestamp-negotiation-05

Abstract

A number of TCP enhancements in diverse fields as congestion control, loss recovery or side-band signaling could be improved by allowing both ends of a TCP session to interpret the value carried in the Timestamp option. Further enhancements are enabled by changing the receiver side processing of timestamps in the presence of Selective Acknowledgements.

This document updates RFC1323 and specifies a backward-compatible method for negotiating for additional capabilities for the Timestamp option, and lists a number of benefits and drawbacks of this approach.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 25, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	6
3. Overview of the TCP Timestamp Option . . . . .	7
4. Extended Timestamp Capabilities . . . . .	8
4.1. Description . . . . .	8
4.2. Timestamp echo update for Selective Acknowledgments . . . . .	9
5. Timestamp capability signaling and negotiation . . . . .	10
5.1. Capability Flags . . . . .	10
5.2. Timestamp clock interval encoding . . . . .	12
5.3. Negotiation error detection and recovery . . . . .	12
5.4. Interaction with Selective Acknowledgment . . . . .	14
5.4.1. Interaction with the Retransmission Timer . . . . .	15
5.4.2. Interaction with the PAWS test . . . . .	16
5.5. Discussion . . . . .	16
6. Acknowledgements . . . . .	17
7. Updates to Existing RFCs . . . . .	17
8. IANA Considerations . . . . .	18
9. Security Considerations . . . . .	19
10. References . . . . .	19
10.1. Normative References . . . . .	19
10.2. Informative References . . . . .	19
Appendix A. Possible use cases . . . . .	21
A.1. Timestamp clock rate exposure . . . . .	21
A.2. Early spurious retransmit detection . . . . .	22
A.3. Early lost retransmission detection . . . . .	23
A.4. Integrity of the Timestamp value . . . . .	24
A.5. Disambiguation with slow Timestamp clock . . . . .	25
A.6. Masked timestamps as segment digest . . . . .	26
Appendix B. Open Issues . . . . .	27
Appendix C. Revision history . . . . .	27
Authors' Addresses . . . . .	29

## 1. Introduction

The Timestamp option originally introduced in [RFC1323] was designed to support only two very specific mechanisms, round trip time measurement (RTTM), and protection against wrapped sequence numbers (PAWS), assuming a particular TCP algorithm (Reno). The current semantics inhibit the use of the Timestamp option for other uses. Taking advantage of developments since TCP Reno, in particular Selective Acknowledgements (SACK) [RFC2018] allow different semantics, which in turn enable new uses for the Timestamp option, either for timing purposes (e.g. one-way delay variation measurement in the context of congestion control), or as unique token (e.g. for improved loss recovery).

This specification defines a protocol for the two ends of a TCP session to negotiate alternative semantics of the Timestamp option fields they will exchange during the rest of the session. It updates RFC1323 but it is backwards compatible with implementations of RFC1323 Timestamp options, and allows gradual deployment.

The RFC1323 timestamp protocol presents the following problems when trying to extend it for alternative uses:

a. Unclear meaning of the value in a timestamp.

- \* A timestamp value (TSval) as defined in [RFC1323] is deliberately only meaningful to the end that sends it. The other end is merely meant to echo the value without understanding it. This is fine if one end is trying to measure two-way delay (round trip time). However, to measure one-way delay variation, timestamps from both ends need to be compared by one end, which needs to relate the values in timestamps from both ends to a notion of the passage of time that both ends share.

b. No control over which timestamp to echo.

- \* A host implementing [RFC1323] is meant to echo the timestamp value of the most recent in-order segment received. This was fine for TCP Reno, but it is not the best choice for TCP sessions using selective acknowledgement (SACK) [RFC2018].
- \* A [RFC1323] host is meant to echo the timestamp value of the earliest unacknowledged segment, e.g. if a host delays ACKs for one segment, it echoes the first timestamp not the second. It is desirable to include delay due to ACK withholding when a host is conservatively measuring RTT. However, is not useful to include the delay due to ACK withholding when measuring

one-way delay variation.

c. Alternative protection against wrapped sequence numbers.

- \* [RFC1323] also points out that the timestamps it specifies will always strictly monotonically increase in each window so they can be used to protect against wrapped sequence numbers (PAWS). If the endpoints negotiate an alternative timestamp scheme in which timestamps may not monotonically increase per window, then it needs to be possible to negotiate alternative protection against wrapped sequence numbers.

To solve these problems this specification changes the wire protocol of the TCP timestamp option in two main ways:

1. It updates [RFC1323] to add the ability to negotiate the semantics of timestamp options. The initiator of a TCP session starts the negotiation in the TSsecr field in the first <SYN>, which is currently unused. This specification defines the semantics of the TSsecr field in a segment with the SYN flag set. A version number is included to allow further extension of capability negotiation in future.
2. A version independent ability to mask a specified number of the lower significant bits of the timestamp values is present. These masked bits are not considered for timestamp calculations, or in an algorithm to protect against wrapped sequence numbers. Future extensions can thereby change the timestamp signaling without changing the modified treatment on the receiver side.
3. It updates [RFC1323] to define version 0 of timestamp capabilities to include:
  - \* the duration in seconds of a tick of the timestamp clock using a time interval representation defined in [I-D.trammell-tcpm-timestamp-interval].
  - \* agreement that both ends will echo the timestamp on the most recently received segment, rather than the one that would be echoed by an [RFC1323] host. There is no specific option to request this behavior, however it is implied by successful negotiation of both SACK and timestamp capabilities.

With this new wire protocol, a number of new use-cases for the TCP timestamp option become possible. Appendix A gives some examples. Further extensions might be required in future. Two possible ways to extend the negotiation capabilities are mentioned, one maintaining some of the semantics specified herein, and a incompatible extension

to allow for other semantics.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The reader is expected to be familiar with the definitions given in [RFC1323].

Further terminology used within this document:

### Timestamp option

This refers to the entire TCP timestamp option, including both TSval and TSecr fields.

### Timestamp capabilities

Refers only to the values and bits carried in the TSecr field of <SYN> and <SYN,ACK> segments during a TCP handshake. For signaling purposes, the timestamp capabilities are sent in clear with the <SYN> segment, and in an encoded form (see Section 5 for details) in the <SYN,ACK> segment.

### 3. Overview of the TCP Timestamp Option

The TCP Timestamp option (TSopt) provides timestamp echoing for round-trip time (RTT) measurements. TSopt is widely deployed and activated by default in many systems. [RFC1323] specifies TSopt the following way:

Kind: 8

Length: 10 bytes

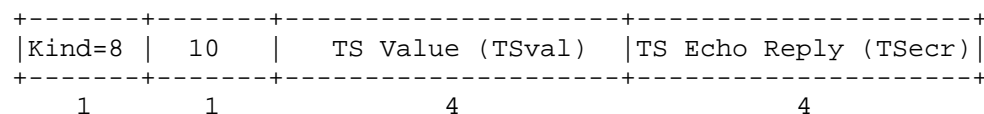


Figure 1: RFC1323 TSopt

"The Timestamps option carries two four-byte timestamp fields. The Timestamp Value field (TSval) contains the current value of the timestamp clock of the TCP sending the option.

The Timestamp Echo Reply field (TSecr) is only valid if the ACK bit is set in the TCP header; if it is valid, it echos a timestamp value that was sent by the remote TCP in the TSval field of a Timestamps option. When TSecr is not valid, its value must be zero. The TSecr value will generally be from the most recent Timestamp option that was received; however, there are exceptions that are explained below.

A TCP may send the Timestamps option (TSopt) in an initial <SYN> segment (i.e., segment containing a SYN bit and no ACK bit), and may send a TSopt in other segments only if it received a TSopt in the initial <SYN> segment for the connection."

The comparison of the timestamp in the TSecr field to the current timestamp clock gives an estimation of the two-way delay (RTT). With [RFC1323] the receiver is not supposed to interpret the TSval field for timing purposes, e.g. one-way delay variation measurements, but only to echo the content in the TSecr field. [RFC1323] specifies various cases when more than one timestamp is available to echo. The only property exposed to a receiver is a strict monotonic increase in value, for use with the protection against wrapped sequence numbers (PAWS) test. The approach taken by [RFC1323] is not always be the best choice, i.e. when the TCP Selective Acknowledgment option (SACK) is used in conjunction on the same session.

## 4. Extended Timestamp Capabilities

### 4.1. Description

Timestamp values are carried in each segment if negotiated for. However, the content of these values is to be treated as an unmutable and largely uninterpreted entity by the receiver. The timestamp negotiation should allow for following criteria:

- o Allow to state timing information explicitly during the initial handshake, avoiding the proliferation of ad-hoc heuristics to determine this information via some other means. Heuristics that simply assume a specific timestamp clock intervals, or try to learn the clock interval used by the partner during a training phase extending beyond the initial handshake can thereby avoided. This is discussed further in [I-D.trammell-tcpm-timestamp-interval].
- o Indicate the (approximate) timestamp clock interval used by the sender in a wide range. The longest interval should be around 10 seconds, while the shorted interval should allow unique timestamps per segment, even at extremely high link speeds. A negotiation-method-independent representation for timestamp intervals is given in [I-D.trammell-tcpm-timestamp-interval].
- o Allow for timestamps that are not directly related to real time (i.e. segment counting, or use of the timestamp value as a true extension of sequence numbers).
- o Provide means to prevent or at least detect tampering with the echoed timestamp value, allowing for basic integrity and consistency checks.
- o Allow for future extensions that may use some of the timestamp value bits for other signaling purposes during the remainder of the session.
- o Signaling must be backwards compatible with existing TCP stacks implementing basic [RFC1323] timestamps. Current methods for timestamp value generation must be supported.
- o Allow for a means to disambiguate between retransmitted and delayed <SYN> segments.
- o Cater for broken implementations of [RFC1323], that either send a non-zero TSsecr value in the initial <SYN>, or a zero TSsecr value in <SYN,ACK>.

- o Provide flexibility to extend the negotiation protocol. Backwards-compatible and incompatible extensions of using timestamps should be available.

#### 4.2. Timestamp echo update for Selective Acknowledgments

In [RFC1323], timing information is only considered in relation to calculating a (conservative) estimate of the round trip time, in order to arrive at a reasonable retransmission timeout (RTO). A retransmission timeout is a very expensive event in TCP, in terms of lost throughput and other metrics. For this reason, a receiver had to follow special rules in what timestamp to echo. This was to never underestimate the actual RTT, even during periods of loss or reordering on either the forward or return path. No other explicit signal could convey the presence of such events back to the sender at the time [RFC1323] was defined. Therefore a receiver had to make sure than at best, the timestamp of the last in-sequence segment would be echoed to the sender.

Receivers conforming to [RFC1323] are required to only reflect the timestamp of the last segment that was received in order, or the timestamp of the last not yet acknowledged segment in the case of delayed acknowledgments.

When selective acknowledgment (SACK) is enabled on a session, the presence of a SACK option will explicitly signal reordering or loss to the sender. This information can be used to suspend the calculation of updated RTT estimates. As the SACK option will be present in multiple ACKs, this also prevents increasing RTT artificially when some of the ACKs, indicating loss, are dropped on the return path.

A receiver supporting the timestamp negotiation mechanism defined in this document MUST immediately reflect the value of TSval in the segment triggering an ACK, when the same session also supports SACK.

The rules to update the state variable TS.recent remain the identical to [RFC1323], and TS.recent must be evaluated when performing the PAWS test on the receiver side.

By this change of semantics when using the timestamps and selective acknowledgments [RFC2018] in the same session, enhancements in loss recovery are possible by removing any remaining retransmission and acknowledgment ambiguity. See Appendix A for a more detailed discussion. Through the modification to the handling of which timestamp to echo in the receiver, timestamps fulfill the properties of the "token", as described in [I-D.sabatini-tcp-sack].

## 5. Timestamp capability signaling and negotiation

In order to signal the supported capabilities, both the sender and the receiver will independently generate a timestamp capability negotiation field, as indicated below. The TSecr value field of the [RFC1323] TSOpt is overloaded with the following flags and fields during the initial <SYN> and <SYN,ACK> segments. The connection initiator will send the timestamp capabilities in plain, as with [RFC1323] the TSecr is not used in the initial <SYN>. The receiver will XOR the local timestamp capabilities with the TSval received from the sender and send the result in the TSecr field. The initiating host of a session with timestamp capability negotiation has to keep minimal state to decode the returned capabilities XOR'ed with the sent TSval.

### 5.1. Capability Flags

Kind: 8

Length: 10 bytes

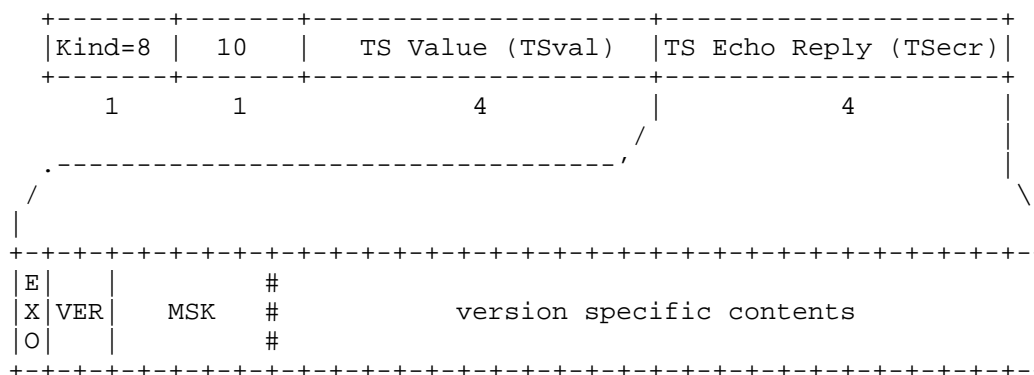


Figure 2: Timestamp Capability flags

Common fields to all versions:

#### EXO - Extended Options (1 bit)

Indicates that the sender supports extended timestamp capabilities as defined by this document, and MUST be set to one by a compliant implementation. This flag also enables the immediate echoing of the TSval with the next ACK, if both timestamp capabilities and selective acknowledgement [RFC2018] are successful negotiated during the initial handshake (see Section 4.2, and Section 5.4). This change in semantics is independent of the version in the signaled timestamp

capabilities.

VER - Version (2 bits)

Version of the capabilities fields definition. This document specifies codepoint 0 (00b). With the exception of the immediate mirroring - simplifying the receiver side processing - and the masking of some LSB bits before performing the Protection Against Wrapped Sequence Numbers (PAWS) test, hosts must not interpret the received timestamps and not use a timestamp value as input into advanced heuristics, if the version received is not supported. This is an identical requirement as with current [RFC1323] compliant implementations.

The lower 3 octets of the timestamp capability flags MUST be ignored if an unsupported version is received. It is expected, that a host will implement at least version 0. A receiver MUST respond with the appropriate (equal or version 0) version when responding to a new session request.

MSK - Mask Timestamps (5 bits)

The MaSK field indicates how many least significant bits should be excluded by the receiver, before further processing the timestamp (i.e. PAWS, or for timing purposes). The unmasked portion of a TSval has to comply with the constraints imposed by [RFC1323] on the generation of valid timestamps, e.g. must be monotone increasing between segments, and strict monotone increasing for each TCP window.

Note that this does not impact the reflected timestamp in any way - TSecr will always be equal to an appropriate TSval. This field MUST be present in all future version of timestamp capability fields. A value of 31 (all bits set) MUST be interpreted by a receiver that the full TSval is to be ignored by any legacy heuristics, e.g. disabling PAWS. For PAWS to be effective, at least two not masked bits are required to discriminate between an increase (and roll-over) versus outdated segments.

## 5.2. Timestamp clock interval encoding

Kind: 8

Length: 10 bytes

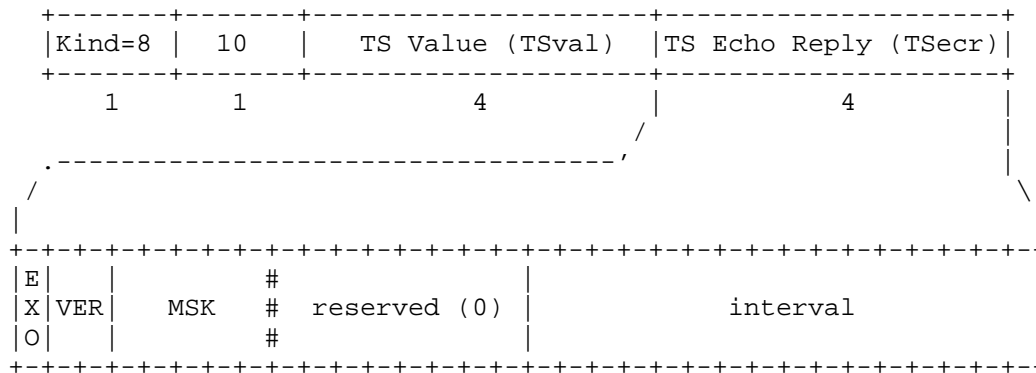


Figure 3: Timestamp Capability flags - version 0

reserved (8 bits)

Reserved for future use, and MUST be zero ("0") with version 0. If timestamp capabilities are received with version set to 0, but some of these bits set, the receiver MUST ignore the extended options field and react as if the TSecr was zero (compatibility mode).

interval (16 bits)

The interval of the timestamp clock, as defined in [I-D.trammell-tcpm-timestamp-interval].

## 5.3. Negotiation error detection and recovery

During the initial TCP three-way handshake, timestamp capabilities are negotiated using the TSecr field. Timestamp capabilities MAY only be negotiated in TSecr when the SYN bit is set. A host detects the presence of timestamp capability flags when the EXO bit is set in the TSecr field of the received <SYN> segment. When receiving a session request (<SYN> segment with timestamp capabilities), a compliant TCP receiver is required to XOR the received TSval with the receivers timestamp capabilities. The resulting value is then sent in the <SYN,ACK> response.

To support these design goals stated in Section 4, only the TSecr field in the initial <SYN> can be used directly. The response from

the receiver has to be encoded, since no unused field is available in the <SYN,ACK>. The most straightforward encoding is a XOR with a value that is known to the sender. Therefore, the receiver also uses TSecr to indicate its capabilities, but calculates the XOR sum with the received TSval. This allows the receiver to remain stateless and functionality like SYN Cache (see [RFC4987]) can be maintained with no change.

If a sender has to retransmit the <SYN>, this encoding also allows to detect which segment was received and responded to. This is possible by changing the timestamp clock offset between retransmissions in such a way, that the decoding on the sender side would result in an invalid timestamp capability negotiation field (e.g. some RES bits are set). If the sender does not require the capability to differentiate which <SYN> was received, the timestamp clock offset for each new <SYN> can be set in such a way, that the TSopt of the <SYN> is identical for each retransmission.

As a receiver MAY report back a zero value at any time, in particular during the <SYN,ACK>, the sender is slightly constrained in its selection of an initial Timestamp value. The Timestamp value sent in the <SYN> should be selected in such a way, that it does not resemble a valid Timestamp capabilities field. One approach to ensure this property is that the sender makes sure that at least one bit of the RES field is set. This prevents a compliant sender to erroneously detect a compliant receiver, if the returned TSecr value is zero.

A host initiating a TCP session must verify if the partner also supports timestamp capability negotiation and a supported version, before using enhanced algorithms. Note that this change in semantics does not necessarily change the signaling of timestamps on the wire after initial negotiation.

To mitigate the effect from misbehaving TCP senders appearing to negotiate for timestamp capabilities, a receiver MUST verify that one specific bit (EXO) is set, and any reserved bits (currently 8, RES field) are cleared. This limits the chance for a receiver to mistakenly negotiate for version 0 capabilities in the presence of a misbehaving sender to around 0.05%. The prevalence of misbehaving senders, and distribution of observed TSecr values, limits this to less than 1 in 6 million. The modifications described in [I-D.ietf-tcpm-1323bis] and implemented in a receiver would further decrease the false negotiation to less than  $10^{-7}$ .

However, as a receiver has to use changed semantics when reflecting TSval also for higher values in the version field, a misbehaving sender negotiating for SACK, but not properly clearing TSecr, may have a 37.5% chance of receiving timestamp values with modified

receiver behavior (from an approximate population of 0.00036% of sessions being started without a cleared TSecr). This may lead to an increased number of spurious retransmission timeouts, putting such a session from a misbehaving TCP sender to a disadvantage.

Once timestamp capabilities are successfully negotiated, the receiver MUST ignore an indicated number of masked, low-order bits, before applying the heuristics defined in [RFC1323]. The monotonic increase of the timestamp value for each new segment could be violated if the full 32 bit field, including the masked bits, are used. This conflicts with the constraints imposed by PAWS.

The presented distribution of the common three fields, EXO, VER and MASK, that MUST be present regardless of which version is implemented in a compliant TCP stack, is a result of the previously mentioned design goals. The lower three octets MAY be redefined freely with subsequent versions of the timestamp capability negotiation protocol. This allows a future version to be implemented in such a way, that a receiver can still operate with the modified behavior, and a minimum amount of processing (PAWS)

#### 5.4. Interaction with Selective Acknowledgment

If both Timestamp capabilities and Selective Acknowledgement options [RFC2018] are negotiated (both hosts send these options in their respective handshake segments), both hosts MUST echo the timestamp value of the last received segment, irrespective of the order of delivery. Note that this is in conflict with [RFC1323], where only the timestamp of the last segment received in sequence is mirrored. As SACK allows discrimination of reordered or lost segments, the reflected timestamp is not required to convey the most conservative information. If SACK indicates lost or reordered packets at the receiver, the sender MUST take appropriate action such as ignoring the received timestamps for calculating the round trip time, or assuming a delayed packet (with appropriate handling). An updated algorithm to calculate the retransmission timeout timer (RTO) is beyond the scope of this document.

The immediate echoing of the last received timestamp value allowed by the simultaneous use of the timestamp option with the SACK option enables enhancements to improve loss recovery, round trip time (RTT) and one-way delay (OWD) variation measurements (see Appendix A) even during loss or reordering episodes. This is enabled by removing any retransmission ambiguity using unique timestamps for every retransmission, while simultaneously the SACK option indicates the ordering of received segments even in the presence of ACK loss or reordering.

For legacy applications of the timestamp option such as RTTM and PAWS, the presence of the SACK option gives a clear indication of loss or reordering. Under these circumstances, RTTM should not be invoked even under [RFC1323], but often is, due to separate handling of timestamp and SACK options).

The use of RTT and OWD measurements during loss episodes is an open research topic. A sender has to accommodate for the changed timestamp semantics in order to maintain a conservative estimate of the Retransmission Timer as defined in [RFC6298], when a TCP sender has negotiated for an immediate reflection of the timestamp triggering an ACK (i.e. both timestamp capability negotiation and Selective Acknowledgements are enabled for the session). As the presence of a SACK option in an ACK signals an ongoing reordering or loss episode, timestamps conveyed in such segments MUST NOT be used to update the retransmission timeout. Also note that the presence of a SACK option alleviates the need of the receiver to reflect the last in-order timestamp, as lost ACKs can no longer cause erroneous updates of the retransmission timeout.

#### 5.4.1. Interaction with the Retransmission Timer

The above stated rule, to ignore timestamps as soon as a SACK option is present, is fully consistent with the guidance given in [RFC1323], even though most implementations skip over such an additional verification step in the presence of the SACK option.

To address the additional delay imposed by delayed ACKs, a compliant sender SHOULD modify the update procedure when receiving normal, in-sequence ACKs that acknowledge more than SMSS bytes, so that the input (denoted R in [RFC6298]) is calculated as

$$R = RTT * ( 1 + 1/(cwnd/smss) )$$

If RTT (as measured in units of the timestamp clock) is smaller than the congestion window measured in full sized segments, the above heuristic MAY be bypassed before updating the retransmission timeout value.

#### 5.4.2. Interaction with the PAWS test

The PAWS test as defined in [RFC1323] requires constant monotonic increasing values at the receiver side. As TS.Recent is no longer used to track which timestamp to echo, this variable can be reused. Instead of tracking the timestamp sent in the most recent ACK, a more strict update rule could be used:

"For example, we might save the timestamp from the segment that last advanced the left edge of the receive window, i.e., the most recent in-sequence segment."

TS.Recent is only to be updated whenever the left window advances, but no longer has to consider delayed ACKs.

#### 5.5. Discussion

RTT and OWD variation during loss episodes is not deeply researched. Current heuristics ([RFC1122], [RFC1323], Karn's algorithm [RFC2988]) explicitly exclude (and prevent) the use of RTT samples when loss occurs. However, solving the retransmission ambiguity problem - and the related reliable ACK delivery problem - would enable new functionality to improve TCP processing. Also, having an immediate echo of the last received timestamp value would enable new research to distinguish between corruption loss (assumed to have no RTT / OWD impact) and congestion loss (assumed to have RTT / OWD impact). Research into this field appears to be rather neglected, especially when it comes to large scale, public internet investigations. Due to the very nature of this, passive investigations without signals contained within the headers are only of limited use in empirical research.

Retransmission ambiguity detection during loss recovery would allow an additional level of loss recovery control without reverting to timer-based methods. As with the deployment of SACK, separating "what" to send from "when" to send it could be driven one step further. In particular, less conservative loss recovery schemes which do not trade principles of packet conservation against timeliness, require a reliable way of prompt and best possible feedback from the receiver about any delivered segment and their ordering. [RFC2018] SACK alone goes quite a long way, but using timestamp information in addition could remove any ambiguity. However, the current specs in [RFC1323] make that use impossible, thus a modified semantic (receiver behavior) is a necessity.

A change in signaling with immediate timestamp value echoes would however break some legacy, per-packet RTT measurements. The reason is, that delayed ACKs would not be covered. Research has shown, that

per-packet updates of the RTT estimation (for the purpose of calculating a reasonable RTO value) are only of limited benefit (see [Path99], and [PH04]). This is the most serious implication of the proposed signaling scheme with directly echoing the timestamp value of the segment triggering the ACK, when the SACK options is also negotiated for. Even when using the directly reflected timestamp values in an unmodified RTT estimator, the immediate impact would be limited to causing premature RTOs when the sending rate suddenly drops below two segments per RTT. That is, assuming the receiver implements delayed ACK and sending one ACK for every other data segment received. If the receiver has also D-SACK [RFC2883] enabled, such premature RTOs can be detected and mitigated by the sender (for example, by increasing minRTO for low bandwidth flows).

Allowing timestamps to play a more important role in TCP signaling also gives rise to concerns. When the timestamp is used for congestion control purposes, this gives an incentive for malicious receivers to reflect tampered timestamps. During the early phases of the introduction of Cubic, such modifications were shown to result in unfair advantages to malicious receivers, that selectively alter the reflected timestamp values (see [CUBIC]). For that very reason, this document introduces the explicit possibility to include a signal in the timestamp values that is excluded from any processing by the receiver. A sender can then decide how to make use of this capability, e.g. for use as additional security information, improvements of loss recovery or other, yet unknown, means.

## 6. Acknowledgements

The authors would like to thank Dragana Damjanovic for some initial thoughts around Timestamps and their extended potential use.

We would like to thank Bob Briscoe for his insightful comments, and the gratuitous donation of text, that have resulted in a substantially improved document.

We further want to thank Michael Welzl for his input and discussion.

## 7. Updates to Existing RFCs

Care has been taken to make sure the updates in this specification can be deployed incrementally.

Updates to existing [RFC1323] implementations are only REQUIRED if they do not clear the TSecr value in the initial <SYN> segment. This is a misinterpretation of [RFC1323] and may leak data anyway (see

[I-D.ietf-tcpm-tcp-security]). Also see [I-D.ietf-tcpm-1323bis], as this stipulates, that the TSval sent in a <RST> should be zeroed, further reducing the chance for a false positive. It is expected, that these changes are implemented in stacks making use of timestamp negotiation. Otherwise, there will be no need to update an RFC1323-compliant TCP stack unless the timestamp capabilities negotiation is to be used.

Implementations compliant with the definitions in this document shall be prepared to encounter misbehaving senders, that don't clear TSecr in their initial <SYN>. It is believed, that checking the reserved bits to be all zero provides enough protection against misbehaving senders.

In the unlikely case of an erroneous negotiation of timestamp capabilities between a compliant receiver, and a misbehaving sender, the proposed semantic changes will trigger a higher rate of spurious retransmissions, while time-based heuristics on the receiver side may further negatively impact congestion control decisions. Overall, misbehaving receivers will suffer from self-inflicted reductions in TCP performance.

## 8. IANA Considerations

With this document, the IANA is requested to establish a new registry to record the timestamp capability flags defined with future versions (codepoints 1, 2 and 3).

The lower 24 bits (3 octets) of the timestamp capabilities field may be freely assigned in future versions. The first octet must always contain the EXO, VER and MASK fields for compatibility, and the MASK field MUST be set to allow interoperation with a version 0 receiver.

This document specifies version 0 and the use of the last three octets to signal the senders timestamp clock interval to the receiver.

## 9. Security Considerations

The algorithm presented in this paper shares security considerations with [RFC1323] (see [I-D.ietf-tcpm-tcp-security]).

An implementation can address the vulnerabilities of [RFC1323], by dedicating a few low-order bits of the timestamp fields for use with a (secure) hash, that protects against malicious modification of returned timestamp value by the receiver. A MASK field has been provided to explicitly notify the receiver about that alternate use of low-order bits. This allows the use of timestamps for purposes requiring higher integrity and security while allowing the receiver to extract useful information nevertheless.

## 10. References

### 10.1. Normative References

- [I-D.trammell-tcpm-timestamp-interval]  
Scheffenegger, R., Kuehlewind, M., and B. Trammell,  
"Exposure of Time Intervals for the TCP Timestamp Option",  
draft-trammell-tcpm-timestamp-interval-00 (work in  
progress), October 2012.
- [RFC1323] Jacobson, V., Braden, B., and D. Borman, "TCP Extensions  
for High Performance", RFC 1323, May 1992.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP  
Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate  
Requirement Levels", BCP 14, RFC 2119, March 1997.

### 10.2. Informative References

- [BSD10] Hayes, D., "Timing enhancements to the FreeBSD kernel to  
support delay and rate based TCP mechanisms", Feb 2010, <[http://caia.swin.edu.au/reports/100219A/  
CAIA-TR-100219A.pdf](http://caia.swin.edu.au/reports/100219A/CAIA-TR-100219A.pdf)>.
- [CUBIC] Rhee, I., Ha, S., and L. Xu, "CUBIC: A New TCP-Friendly  
High-Speed TCP Variant", Feb 2005, <[http://  
citeseerx.ist.psu.edu/viewdoc/  
download?doi=10.1.1.153.3152&rep=rep1&type=pdf](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.153.3152&rep=rep1&type=pdf)>.
- [Cho08] Cho, I., Han, J., and J. Lee, "Enhanced Response Algorithm  
for Spurious TCP Timeout (ER-SRTO)", Jan 2008, <<http://>

ubinet.yonsei.ac.kr/v2/publication/hpmn\_papaers/ic/  
2008\_Enhanced%20Response%20Algorithm%20for%20Spurious%  
20TCP.pdf>.

[I-D.blanton-tcp-reordering]

Blanton, E., Dimond, R., and M. Allman, "Practices for TCP  
Senders in the Face of Segment Reordering",  
draft-blanton-tcp-reordering-00 (work in progress),  
February 2003.

[I-D.ietf-tcpm-1323bis]

Borman, D., Braden, R., Jacobson, V., and R.  
Scheffenegger, "TCP Extensions for High Performance",  
draft-ietf-tcpm-1323bis-04 (work in progress),  
August 2012.

[I-D.ietf-tcpm-anumita-tcp-stronger-checksum]

Biswas, A., "Support for Stronger Error Detection Codes in  
TCP for Jumbo Frames",  
draft-ietf-tcpm-anumita-tcp-stronger-checksum-00 (work in  
progress), May 2010.

[I-D.ietf-tcpm-tcp-security]

Gont, F., "Survey of Security Hardening Methods for  
Transmission Control Protocol (TCP) Implementations",  
draft-ietf-tcpm-tcp-security-03 (work in progress),  
March 2012.

[I-D.sabatini-tcp-sack]

Sabatini, A., "Highly Efficient Selective Acknowledgement  
(SACK) for TCP", draft-sabatini-tcp-sack-01 (work in  
progress), August 2012.

[Linux]

Sarolahti, P., "Linux TCP", Apr 2007,  
<<http://www.cs.clemson.edu/~westall/853/linuxtcp.pdf>>.

[PH04]

Eckstroem, H. and R. Ludwig, "The Peak-Hopper: A New End-  
to-End Retransmission Timer for Reliable Unicast  
Transport", Apr 2004, <[citeseerx.ist.psu.edu/viewdoc/  
download?doi=10.1.1.76.2748&rep=rep1&type=pdf](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.2748&rep=rep1&type=pdf)>.

[Path99]

Allman, M. and V. Paxson, "On Estimating End-to-End  
Network Path Properties", Sep 1999,  
<<http://www.icir.org/mallman/papers/estimation.ps>>.

[RFC1122]

Braden, R., "Requirements for Internet Hosts -  
Communication Layers", STD 3, RFC 1122, October 1989.

- [RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.
- [RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.
- [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.
- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.
- [RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC 6013, January 2011.
- [RFC6247] Eggert, L., "Moving the Undeployed TCP Extensions RFC 1072, RFC 1106, RFC 1110, RFC 1145, RFC 1146, RFC 1379, RFC 1644, and RFC 1693 to Historic Status", RFC 6247, May 2011.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

## Appendix A. Possible use cases

### A.1. Timestamp clock rate exposure

Today, each TCP host may use an arbitrary, locally defined clock source to derive the timestamp value from. Even though only a handful of typically used clock rates are implemented in common TCP stacks, this does not guarantee that any future stack will choose the same clock rate. This poses a problem for current state of the art heuristics, which try to determine the senders timestamp clock rate by pure passive observation of the TCP stream, and affects both advanced heuristics in the partner host of a TCP session, or arbitrarily located passive observation points to estimate TCP session parameters.

The proposed mechanism would reveal this information explicitly, even though other environmental factors, such as the operation of a TCP stack in a virtualized environment, may result in some deviations in the actually used clock rate.

High-speed and real-time stacks would be expected to operate with higher clock rates, while the observed variance in (known) timestamp clock vs. reference clock could help in determining between physical and virtual end hosts, for example.

#### A.2. Early spurious retransmit detection

Using the provided timestamp negotiation scheme, clients utilizing slow running timestamp clocks can set aside a small number of least significant bits in the timestamps. These bits can be used to differentiate between original and retransmitted segments, even within the same timestamp clock tick (i.e. when RTT is shorter than the TCP timestamp clock interval). It is recommended to use only a single bit (mask = 1), unless the sender can also perform lost retransmission detection. Using more than 2 bits for this purpose is discouraged due to the diminishing probability of losing retransmitted packets more than one time. A simple scheme could send out normal data segments with the so masked bits all cleared. Each advance of the timestamp clock also clears those bits again. When a segment is retransmitted without the timestamp clock increasing, these bits increased by one for each consecutive retry of the same segment, until the maximum value is reached. Newly sent segments (during the same clock interval) should maintain these bits, in order to maintain monotonically increasing values, even though compliant end hosts do not require this property. This scheme maintains monotonically increasing timestamp values - including the masked bits. Even without negotiating the immediate mirroring of timestamps (done by simultaneously doing timestamp capabilities negotiation, and selective acknowledgments), this extends the use of the Eifel Detection [RFC3522] and Eifel Response [RFC4015] algorithm to detect and react to spurious retransmissions under all circumstances. Also, currently experimental schemes such as ER-SRTO [Cho08] could be deployed without requiring the receiver to explicitly support that capability.

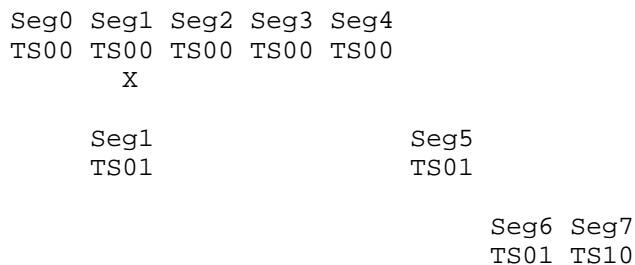


Figure 4: timestamp for spurious retransmit detection

Masked bits are the 2nd digit, the timestamp value is represented by the first digit. The timestamp clock "ticks" between segment 6 and 7.

#### A.3. Early lost retransmission detection

During phases where multiple segments in short succession (but not necessarily successive segments) are lost, there is a high likelihood that at least one segment is retransmitted, while the cause of loss (i.e. congestion, fading) is still persisting. The best current algorithms can recover such a lost retransmission with a few constraints, for example, that the session has to have at least DupThresh more segments to send beyond the current recovery phase. During loss recovery, when a retransmission is lost again, currently the timestamp can also not be used as means of conveying additional information, to allow more rapid loss recovery while maintaining packet conservation principles. Only the timestamp of the last segment preceding the continuous loss will be reflected. Using the extended timestamp option negotiation together with selective acknowledgements, the receiver will immediately reflect the timestamp of the last seen segment. Using both SACK and TS information in conjunction with each other, a sender can infer the exact order in which original and retransmitted segments are received. This allows faster recovery from lost retransmissions while maintaining the principle of packet conservations and avoiding costly retransmission timeouts.

The implementation can be done in combination with the masked bit approach described in the previous paragraph, or without. However, if the timestamp clock interval is lower than  $1/2$  RTT, both the original and the retransmitted segment may carry an identical timestamp. If the sender cannot discriminate between the original and the retransmitted segments, it must refrain from taking any action before such a determination can be made.

In this example, masked bits are used, with a simple marking method. As the timestamp value of the retransmission itself is already different from the original segments, such an additional discrimination would not strictly be required here. The timestamp clock ticks in the first digit and the dupthresh value is 3.

Seg0	Seg1	Seg2	Seg3	Seg4	Seg5	Seg6	Seg7
TS00	TS00	TS00	TS10	TS10	TS10	TS10	TS20
	X	X	X	*			
Seg1	Seg2	Seg3	Seg4				
TS21	TS30	TS30	TS30				
X							
Seg1						Seg8	Seg9
TS31						TS31	TS40

Figure 5: timestamp under loss

If Seg1,TS00 is lost twice, and Seg4,TS10 is also lost, the sender could resend Seg1 once more after observing dupthresh number of segments sent after the first retransmission of Seg1 being received (ie, when Seg4 is SACKed). However, there is an ambiguity between retransmitted segments and original segments, as the sender cannot know, if a SACK for one particular segment was due to the retransmitted segment, or a delayed original segment. The timestamp value will not help in this case, as per RFC1323 it will be held at TS00 for the entire loss recovery episode. Therefore, currently a sender has to assume that any SACKed segments may be due to delayed original sent segments, and can only resolve this conflict by injecting additional, previously unsent segments. Once dupthresh newly injected segments are SACKed, continuous loss (and not further delay) of Seg1 can safely be assumed, and that segment be resent. This approach is conservative but constrained by the requirement that additional segments can be sent, and thereby delayed in the response.

With the simultaneous use of timestamp extended options together with selective acknowledgments, the receiver would immediately reflect back the timestamp of the last received segment. This allows the sender to discriminate between a SACK due to a delayed Seg4,TS10, or a SACK because of Seg4,TS30. Therefore, the appropriate decision (retransmission of Seg1 once more, or addressing the observed reordering/delay accordingly [I-D.blanton-tcp-reordering]) can be taken with high confidence.

#### A.4. Integrity of the Timestamp value

If the timestamp is used for congestion control purposes, an incentive exists for malicious receivers to reflect tampered timestamps, as demonstrated with some exploits [CUBIC].

One way to address this is to not use timestamp information directly, but to keep state in the sender for each sent segment, and track the round trip time independent of sent timestamps. Such an approach has

the drawback, that it is not straightforward to make it work during loss recovery phases for those segments possibly lost (or reordered). In addition there is processing and memory overhead to maintain possibly extensive lists in the sender that need to be consulted with each ACK. Despite these drawbacks, this approach is currently implemented due to lack of alternatives (see [Linux], and [BSD10]).

The preferred approach is that the sender MAY choose to protect timestamps from such modifications by including a fingerprint (secure hash of some kind) in some of the least significant bits. However, doing so prevents a receiver from using the timestamp for other purposes, unless the receiver has prior knowledge about this use of some bits in the timestamp value. Furthermore, strict monotonic increasing values are still to be maintained. That constraint restricts this approach somewhat and limits or inhibits the use of timestamp values for direct use by the receiver (i.e. for one-way delay variation measurement, as the hash bits would look like random noise in the delay measurement).

#### A.5. Disambiguation with slow Timestamp clock

In addition, but somewhat orthogonal to maintaining timestamp value integrity, there is a use case when the sender does not support a timestamp clock interval that can guarantee unique timestamps for retransmitted segments. This may happen whenever the TCP timestamp clock interval is higher than the round-trip time of the path. For unambiguously identifying regular from retransmitted segments, the timestamp must be unique for otherwise identical segments. Reserving the least significant bits for this purpose allows senders with slow running timestamp clocks to make use of this feature. However, without modifying the receiver behavior, only limited benefits can be extracted from such an approach. Furthermore the use of this option has implications in the protection against wrapped sequence numbers (PAWS - [RFC1323]), as the more bits are set aside for tamper prevention, the faster the timestamp number space cycles.

Using Timestamp capabilities to explicitly negotiate mask bits, and set aside a (low) number of least significant bits for the above listed purposes, allows a sender to use more reliable integrity checks. These masked bits are not to be considered part of the timestamp value, for the purposes described in [RFC1323] (i.e. PAWS) and subsequent heuristics using timestamp values (i.e. Eifel Detection), thereby lifting the strict requirement of always monotonically increasing timestamp values. However, care should be taken to not mask too many bits, for the reasons outlined in [RFC1323]. Using a mask value higher than 8 is therefore discouraged.

The reason for having 5 bits for the mask field nevertheless is to allow the implementation of this protocol in conjunction with TCP cookie transaction (TCPCT) extended timestamps [RFC6013]. That allows for nearly a quarter of a 128 bit timestamp to be set aside.

#### A.6. Masked timestamps as segment digest

After making TCP alternate checksums historic (see [RFC6247]), there still remains a need to address increased corruption probabilities when segment sizes are increased (see [I-D.ietf-tcpm-anumita-tcp-stronger-checksum]).

Utilizing a completely masked TSval field allows the sender to include a stronger CRC32, with semantics independent of the fixed TCP header fields. However, such a use would again exclude the use of PAWS on the receiver side, and a receiver would need to know the specifics of the digest for processing. It is assumed, that such a digest would only cover the data payload of a TCP segment. In order to allow disambiguation of retransmissions, a special TSval can be defined (e.g. TSval=0) which bypasses regular CRC processing but allows the identification of retransmitted segments.

The full semantics of such a data-only CRC scheme are beyond the scope of this document, but would require a different version of the timestamp capability. Nevertheless, allowing the full TSval to remain unprocessed by the receiver for the purpose of PAWS even in version 0 could still allow the successful negotiation of sender-side enhancements such as loss recovery improvements (see Appendix A.2, and Appendix A.3).

In effect, the masked portion of the timestamp value represent an unreliable out of band signal channel, that could also be used for other purposes than solely performing timestamp integrity checks (for example, this would allow ER-SRTO algorithms [Cho08]).

## Appendix B. Open Issues

- o The split between this draft and [I-D.trammell-tcpm-timestamp-interval] is cursory; additional separation of timestamp interval export may be necessary.
- o [bht] suggest changing the "versioning" construct to a "capabilities" construct, especially since two bits of versioning might as well be none. The base specification would then define the alternate semantics WRT SACK and could use capabilities to define further semantics.
- o [bht] does it make sense to move masking out of the base spec and into the 8 "unused" bits in "version 0" (in order to get more capabilities bits / "magic bits" to reduce erroneous negotiation)?
- o [bht] does it make sense to define SACK-echo as version/capability independent?

## Appendix C. Revision history

This appendix should be removed by the RFC Editor before publishing this document as a RFC.

00 ... initial draft, early submission to meet deadline.

01 ... refined draft, focusing only on those capabilities that have an immediate use case. Also excluding flags that can be substituted by other means (MIR - synergistic with SACK option only, RNG moved to appendix A, BIA removed and the exponent bias set to a fixed value. Also extended other paragraphs.

02 ... updated document after IETF80 - referrals to "timestamp options" were seen to be ambiguous with "timestamp option", and therefore replaced by "timestamp capabilities". Also, the document was reworked to better align with RFC4101. Removed SGN and increased FRAC to allow higher precision.

03 ... removed references to "opaque" and "transparent". substituted "timestamp clock interval" for all instances of rate. Changed signal encoding to resemble a scale/value approach like what is done with Window Scaling. As added benefit, clock quality can be implicitly signaled, since multiple representations can map to identical time intervals. Added discussion around resilience against broken RFC1323 implementations (Win95, Linux 2.3.41+), which deviate from expected Timestamp signaling behavior.

04 ... removed previous appendix A (range negotiation); minor edit to improve wording; moved Section 6 to the Appendix, and removed covert channels from the potential uses; added some text to discuss future versioning (compatible and incompatible variants); changed document structure; added guidance around PAWS; added pseudo-code examples (probably to be removed again)

05 ... added new Open Issues section, added reference to separate interval draft, removed content on timestamp interval exposure which now appears in the interval draft. Removed pseudocode examples until they can be reworked on finalization of the mechanism, as they refer to fields which have changed / moved to the interval draft.

Authors' Addresses

Richard Scheffenegger  
NetApp, Inc.  
Am Euro Platz 2  
Vienna, 1120  
Austria

Phone: +43 1 3676811 3146  
Email: rs@netapp.com

Mirja Kuehlewind  
University of Stuttgart  
Pfaffenwaldring 47  
Stuttgart 70569  
Germany

Email: mirja.kuehlewind@ikr.uni-stuttgart.de

Brian Trammell  
Swiss Federal Institute of Technology Zurich  
Gloriastrasse 35  
8092 Zurich  
Switzerland

Phone: +41 44 632 70 13  
Email: trammell@tik.ee.ethz.ch



TCPM WG  
Internet Draft  
Intended status: Experimental  
Expires: November 2013

J. Touch  
USC/ISI  
May 23, 2013

A TCP Authentication Option Extension for NAT Traversal  
draft-touch-tcp-ao-nat-05.txt

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on November 23, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Abstract

This document describes an extension to the TCP Authentication Option (TCP-AO) to support its use over connections that pass through network address and/or port translators (NATs/NAPTs). This extension changes the data used to compute traffic keys, but does not alter TCP-AO's packet processing or key generation algorithms.

## Table of Contents

1. Introduction.....	2
2. Conventions used in this document.....	2
3. Background.....	3
4. Extension to Allow NAT Traversal.....	3
5. Intended Use.....	4
6. Security Considerations.....	5
7. IANA Considerations.....	5
8. References.....	5
8.1. Normative References.....	5
8.2. Informative References.....	6
9. Acknowledgments.....	6

## 1. Introduction

This document describes an extension to the TCP Authentication Option (TCP-AO) [RFC5925] called TCP-AO-NAT to support its use in the presence of network address and/or port translators (NAT/NAPT) [RFC2663]. These devices translate the source address and/or the source port number of a TCP connection. TCP-AO without TCP-AO-NAT extensions would be sensitive to these modifications, and would discard authenticated segments.

At least one potential application of TCP-AO-NAT is to support the experimental multipath TCP protocol [RFC6824], which uses multiple IP addresses to support a single TCP transfer.

This document assumes detailed familiarity with TCP-AO [RFC5925]. As a preview, this document focuses on how TCP-AO generates traffic keys, and does not otherwise alter the TCP-AO mechanism or that of its key generation [RFC5926].

## 2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

When used in lower case, these words have their conventional meaning and do not convey the interpretations in RFC-2119.

### 3. Background

TCP-AO generates traffic keys that are specific to a socket pair [RFC5925]. Using the TCP-AO convention (local = source for outgoing segments, destination for incoming segments), the following information is used to create a connection's traffic keys:

- o IP local address
- o IP remote address
- o TCP local port
- o TCP remote port
- o TCP local Initial Sequence Number (ISN)
- o TCP remote Initial Sequence Number (ISN)

Of these fields, the remote ISN is not known for SYN segments, and is excluded from the traffic key used to authenticate them. Otherwise, all fields are used in the traffic keys of all other segments.

NATs and NAPT's (both referred to herein as "NATs", even if port translation is included) would interfere with these uses, because they alter the IP address and TCP port of the endpoint behind the NAT [RFC2663].

### 4. Extension to Allow NAT Traversal

The premise of TCP-AO-NAT is that it might be useful to allow TCP-AO use in the presence of NATs, e.g., to protect client/server communication where clients are behind NATs.

This document describes TCP-AO-NAT, an extension to TCP-AO that enables its use in the presence of NATs. This extension requires no modification to the TCP-AO header or packet processing, and requires no modification to the algorithms used to generate traffic keys [RFC5926]. The change is limited to the data used to generate traffic keys only.

In TCP-AO, "a Master Key Tuple (MKT) describes the TCP-AO properties to be associated with one or more connections" [RFC5925]. This

includes the TCP connection identifier, the TCP option flag (indicating whether TCP options other than TCP-AO are included in the MAC calculation), keying information, and other parameters. TCP-AO-NAT augments the MKT with two additional flags:

- o localNAT
- o remoteNAT

TCP-AO implementations supporting TCP-AO-NAT MUST support both localNAT and remoteNAT flags.

These flags indicate whether a segment's local or remote (respectively) IP address and TCP port are zeroed before MAC calculation, either for creating the MAC to insert (for outgoing segments) or for calculating a MAC to validate against the value in the option. I.e., these would modify TCP-AO processing rules as follows:

- o In TCP-AO-NAT, traffic keys are computed by zeroing the local/remote IP address and TCP port as indicated by the localNAT or remoteNAT flags.
- o In TCP-AO-NAT, MAC values are computed by zeroing the local/remote IP address and TCP port as indicated by the localNAT or remoteNAT flags.

The use of these flags needs to match on both ends of the connection, just as with all other MKT parameters.

## 5. Intended Use

A host MAY use TCP-AO-NAT when it is behind a NAT, as determined using NAT discovery techniques, or when TCP-AO protection is desired but conventional TCP-AO fails to establish connections.

A client behind a NAT MAY set localNAT=TRUE for MKTs supporting TCP-AO-NAT for outgoing connections. A server MAY set remoteNAT=TRUE for MKTs supporting TCP-AO-NAT for incoming connections. Peer-to-peer applications with dual NAT support, e.g., those traversing so-called 'symmetric NATs' [RFC5389], MAY set both localNAT=TRUE and remoteNAT=TRUE for MKTs supporting TCP-AO-NAT bidirectionally. Once these flags are set in an MKT, they affect all connections that match that MKT.

TCP-AO-NAT is intended for use only where coordinated between endpoints for connections that match the shared MKT parameters, as with all other MKT parameters.

Note that TCP-AO-NAT is not intended for use with services transiting application layer gateways (ALGs), i.e., NATs that also translate in-band addresses, such as used in FTP or SIP. TCP-AO-NAT protects the contents of the TCP segments from modification, and would (correctly) interpret with such alterations as an attack on those contents.

## 6. Security Considerations

TCP-AO-NAT does not affect the security of connections that do not set either of the localNAT or remoteNAT flags. Such connections are not affected themselves, and are not affected by segments in other connections that set those flags.

Setting either the localNAT or remoteNAT flags reduces the randomness of the input to the KDF used to generate the traffic keys. The largest impact occurs when using IPv4, which reduces the randomness from 2 IPv4 addresses, 2 ISNs, and both ports down to just the two ISNs when both flags are set. The amount of randomness in the IPv4 addresses and service port is likely to be small, and the randomness of the dynamic port is under debate and should not be considered substantial [RFC6056]. The KDF input randomness is thus expected to be dominated by that of the ISNs, so reducing it by either or both the IPv4 addresses and ports is not expected to have a significant impact.

## 7. IANA Considerations

There are no IANA considerations for this document. This section can be removed upon publication as an RFC.

## 8. References

### 8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5925] Touch, J., A. Mankin, R. Bonica, "The TCP Authentication Option", RFC 5925, Jun. 2010.

## 8.2. Informative References

- [RFC6824] Ford, A., C. Raiciu, M. Handley, O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, Jan. 2013.
- [RFC2663] Srisuresh, P. and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, August 1999.
- [RFC5389] Rosenberg, J., R. Mahy, P. Matthews, D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, Oct. 2008.
- [RFC5926] Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", RFC 5926, June 2010.
- [RFC6056] Larsen, M., F. Gont, "Port Randomization," RFC 6056, Jan. 2011.

## 9. Acknowledgments

This extension was inspired by discussions with Dan Wing.

This document was prepared using 2-Word-v2.0.template.dot.

## Author's Address

Joe Touch  
USC/ISI  
4676 Admiralty Way  
Marina del Rey, CA 90292  
USA

Phone: +1 (310) 448-9151  
Email: touch@isi.edu

