

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 8, 2016

S. Bensley
Microsoft
L. Eggert
NetApp
D. Thaler
P. Balasubramanian
Microsoft
G. Judd
Morgan Stanley
July 7, 2015

Microsoft's Datacenter TCP (DCTCP):
TCP Congestion Control for Datacenters
draft-bensley-tcpm-dctcp-05

Abstract

This memo describes Datacenter TCP (DCTCP), an improvement to TCP congestion control for datacenter traffic. DCTCP uses improved Explicit Congestion Notification (ECN) processing to estimate the fraction of bytes that encounter congestion, rather than simply detecting that some congestion has occurred. DCTCP then scales the TCP congestion window based on this estimate. This method achieves high burst tolerance, low latency, and high throughput with shallow-buffered switches.

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 8, 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	2
2. Terminology	4
3. DCTCP Algorithm	4
3.1. Marking Congestion on the Switch	4
3.2. Echoing Congestion Information on the Receiver	4
3.3. Processing Congestion Indications on the Sender	5
3.4. Handling of SYN, SYN-ACK, RST Packets	7
4. Implementation Issues	7
5. Deployment Issues	8
6. Known Issues	9
7. Implementation Status	10
8. Security Considerations	10
9. IANA Considerations	10
10. Acknowledgements	10
11. References	11
11.1. Normative References	11
11.2. Informative References	11
Authors' Addresses	12

1. Introduction

Large datacenters necessarily need a large number of network switches to interconnect the servers in the datacenter. Therefore, a datacenter can greatly reduce its capital expenditure by leveraging low cost switches. However, low cost switches tend to have limited queue capacities and thus are more susceptible to packet loss due to congestion.

Network traffic in the datacenter is often a mix of short and long flows, where the short flows require low latency and the long flows require high throughput. Datacenters also experience incast bursts,

where many endpoints send traffic to a single server at the same time. For example, this is a natural consequence of MapReduce algorithms. The worker nodes complete at approximately the same time, and all reply to the master node concurrently.

These factors place some conflicting demands on the queue occupancy of a switch:

- o The queue must be short enough that it does not impose excessive latency on short flows.
- o The queue must be long enough to buffer sufficient data for the long flows to saturate the path bandwidth.
- o The queue must be short enough to absorb incast bursts without excessive packet loss.

Standard TCP congestion control [RFC5681] relies on segment loss to detect congestion. This does not meet the demands described above. First, the short flows will start to experience unacceptable latencies before packet loss occurs. Second, by the time TCP congestion control kicks in on the sender, most of the incast burst has already been dropped.

[RFC3168] describes a mechanism for using Explicit Congestion Notification (ECN) from the switch for early detection of congestion, rather than waiting for segment loss to occur. However, this method only detects the presence of congestion, not the extent. In the presence of mild congestion, the TCP congestion window is reduced too aggressively and unnecessarily affects the throughput of long flows.

Datacenter TCP (DCTCP) improvises upon traditional ECN processing by estimating the fraction of bytes that encounter congestion, rather than simply detecting that some congestion has occurred. DCTCP then scales the TCP congestion window based on this estimate. This method achieves high burst tolerance, low latency, and high throughput with shallow-buffered switches.

It is recommended that DCTCP be deployed in a datacenter environment where the endpoints and the switching fabric are under a single administrative domain. Deployment issues around coexistence of DCTCP and conventional TCP, and lack of a negotiating mechanism between sender and receiver, and possible mitigations are also discussed.

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].

3. DCTCP Algorithm

There are three components involved in the DCTCP algorithm:

- o The switch (or other intermediate device on the network) detects congestion and sets the Congestion Encountered (CE) codepoint in the IP header.
- o The receiver echoes the congestion information back to the sender using the ECN-Echo (ECE) flag in the TCP header.
- o The sender reacts to the congestion indication by reducing the TCP congestion window (cwnd).

3.1. Marking Congestion on the Switch

The switch indicates congestion to the end nodes by setting the CE codepoint in the IP header as specified in Section 5 of [RFC3168]. For example, the switch may be configured with a congestion threshold. When a packet arrives at the switch and its queue length is greater than the congestion threshold, the switch sets the CE codepoint in the packet. For example, Section 3.4 of [DCTCP10] suggests threshold marking with a threshold $K > (RTT * C)/7$, where C is the sending rate in packets per second. However, the actual algorithm for marking congestion is an implementation detail of the switch and will generally not be known to the sender and receiver. Therefore, sender and receiver MUST NOT assume that a particular marking algorithm is implemented by the switching fabric.

3.2. Echoing Congestion Information on the Receiver

According to Section 6.1.3 of [RFC3168], the receiver sets the ECE flag if any of the packets being acknowledged had the CE code point set. The receiver then continues to set the ECE flag until it receives a packet with the Congestion Window Reduced (CWR) flag set. However, the DCTCP algorithm requires more detailed congestion information. In particular, the sender must be able to determine the number of sent bytes that encountered congestion. Thus, the scheme described in [RFC3168] does not suffice.

One possible solution is to ACK every packet and set the ECE flag in the ACK if and only if the CE code point was set in the packet being

acknowledged. However, this prevents the use of delayed ACKs, which are an important performance optimization in datacenters.

Instead, DCTCP introduces a new Boolean TCP state variable, DCTCP Congestion Encountered (DCTCP.CE), which is initialized to false and stored in the Transmission Control Block (TCB). When sending an ACK, the ECE flag MUST be set if and only if DCTCP.CE is true. When receiving packets, the CE codepoint MUST be processed as follows:

1. If the CE codepoint is set and DCTCP.CE is false, send an ACK for any previously unacknowledged packets and set DCTCP.CE to true.
2. If the CE codepoint is not set and DCTCP.CE is true, send an ACK for any previously unacknowledged packets and set DCTCP.CE to false.
3. Otherwise, ignore the CE codepoint.

3.3. Processing Congestion Indications on the Sender

The sender estimates the fraction of sent bytes that encountered congestion. The current estimate is stored in a new TCP state variable, DCTCP.Alpha, which is initialized to 1 and MUST be updated as follows:

$$\text{DCTCP.Alpha} = \text{DCTCP.Alpha} * (1 - g) + g * M$$

where

- o g is the estimation gain, a real number between 0 and 1. The selection of g is left to the implementation. See Section 4 for further considerations.
- o M is the fraction of sent bytes that encountered congestion during the previous observation window, where the observation window is chosen to be approximately the Round Trip Time (RTT). In particular, an observation window ends when all the sent bytes in flight at the beginning of the window have been acknowledged.

In order to update DCTCP.Alpha, the TCP state variables defined in [RFC0793] are used, and three additional TCP state variables are introduced:

- o DCTCP.WindowEnd: The TCP sequence number threshold for beginning a new observation window; initialized to SND.UNA.
- o DCTCP.BytesSent: The number of bytes sent during the current observation window; initialized to zero.

- o DCTCP.BytesMarked: The number of bytes sent during the current observation window that encountered congestion; initialized to zero.

The congestion estimator on the sender MUST process acceptable ACKs as follows:

1. Compute the bytes acknowledged (TCP SACK options [RFC2018] are ignored):

$$\text{BytesAked} = \text{SEG.ACK} - \text{SND.UNA}$$

2. Update the bytes sent:

$$\text{DCTCP.BytesSent} += \text{BytesAked}$$

3. If the ECE flag is set, update the bytes marked:

$$\text{DCTCP.BytesMarked} += \text{BytesAked}$$

4. If the sequence number is less than or equal to DCTCP.WindowEnd, then stop processing. Otherwise, the end of the observation window was reached, so proceed to update the congestion estimate as follows:

5. Compute the congestion level for the current observation window:

$$M = \text{DCTCP.BytesMarked} / \text{DCTCP.BytesSent}$$

6. Update the congestion estimate:

$$\text{DCTCP.Alpha} = \text{DCTCP.Alpha} * (1 - g) + g * M$$

7. Determine the end of the next observation window:

$$\text{DCTCP.WindowEnd} = \text{SND.NXT}$$

8. Reset the byte counters:

$$\text{DCTCP.BytesSent} = \text{DCTCP.BytesMarked} = 0$$

Rather than always halving the congestion window as described in [RFC3168], when the sender receives an indication of congestion (ECE), the sender MUST update cwnd as follows:

$$\text{cwnd} = \text{cwnd} * (1 - \text{DCTCP.Alpha} / 2)$$

Thus, when no sent byte experienced congestion, DCTCP.Alpha equals zero, and cwnd is left unchanged. When all sent bytes experienced congestion, DCTCP.Alpha equals one, and cwnd is reduced by half. Lower levels of congestion will result in correspondingly smaller reductions to cwnd.

Just as specified in [RFC3168], TCP should not react to congestion indications more than once every window of data. The setting of the "Congestion Window Reduced" (CWR) bit is also exactly as per [RFC3168].

3.4. Handling of SYN, SYN-ACK, RST Packets

[RFC3168] requires that compliant TCP MUST NOT set ECT on SYN or SYN-ACK packets. [RFC5562] proposes setting ECT on SYN-ACK packets, but maintains the restriction of no ECT on SYN packets. Both these RFCs prohibit ECT in SYN packets due to security concerns regarding malicious SYN packets with ECT set. These RFCs, however, are intended for general Internet use, and do not directly apply to a controlled datacenter deployment. The switching fabric can drop TCP packets that do not have the ECT set in the IP header. If SYN and SYN-ACK packets for DCTCP connections are non-ECT they will be dropped with high probability. For DCTCP connections the sender SHOULD set ECT for SYN, SYN-ACK and RST packets.

4. Implementation Issues

As noted in Section 3.3, the implementation must choose a suitable estimation gain. [DCTCP10] provides a theoretical basis for selecting the gain. However, it may be more practical to use experimentation to select a suitable gain for a particular network and workload. The Microsoft implementation of DCTCP in Windows Server 2012 uses a fixed estimation gain of 1/16.

The implementation must also decide when to use DCTCP. Datacenter servers may need to communicate with endpoints outside the datacenter, where DCTCP is unsuitable or unsupported. Thus, a global configuration setting to enable DCTCP will generally not suffice. DCTCP may be configured based on the IP address of the remote endpoint. Microsoft Windows Server 2012 also supports automatic selection of DCTCP if the estimated RTT is less than 10 msec and ECN is successfully negotiated, under the assumption that if the RTT is low, then the two endpoints are likely on the same datacenter network.

To prevent incast throughput collapse the minimum RTO (MinRTO) used by TCP should be lowered significantly. The default value of MinRTO in Windows is 300 msec which is much greater than the maximum

latencies inside a datacenter. Server 2012 onwards the MinRTO value is configurable allowing values as low as 10 msec on a per subnet or per TCP port basis or even globally. A lower MinRTO value requires corresponding a lower delayed ACK timeout on the receiver. It is recommended that the implementation allow configuration of lower timeouts for DCTCP connections.

In the same vein, it is also recommended that the implementation allow configuration of restarting the cwnd of idle DCTCP connections as described in [RFC5681] since network conditions change rapidly in the datacenter. The implementation can also allow configuration for discarding the value of DCTCP.Alpha after cwnd restart and timeouts.

[RFC3168] forbids the ECN-marking of pure ACK packets because of the inability of TCP to mitigate ACK-path congestion and protocol-wise preferential treatment by routers. However dropping pure ACKs rather than ECN marking them is disadvantageous in traffic scenarios typical in the datacenter. Because of the prevalence of bursty traffic patterns which involve transient congestion, the dropping of ACKS causes subsequent retransmission. It is recommended that the implementation a configuration knob that forces ECT on TCP pure ACK packets.

5. Deployment Issues

DCTCP and conventional TCP congestion control does not coexist well. In DCTCP, the marking threshold is set very low value to reduce queueing delay, thus a relatively small amount of congestion will exceed the marking threshold. During such periods of congestion, conventional TCP will suffer packet losses and quickly scale back cwnd. DCTCP, on the other hand, will use the fraction of marked packets to scale back cwnd. Thus rate reduction in DCTCP will be much lower than that of conventional TCP, and DCTCP traffic will dominate conventional TCP traffic traversing the same link. Hence if the traffic in the datacenter is a mix of conventional TCP and DCTCP, it is recommended that DCTCP traffic be segregated from conventional TCP traffic. [MORGANSTANLEY] describes a deployment that uses IP DSCP bits where AQM is applied to DCTCP traffic, while TCP traffic is managed via drop-tail queueing.

Today's commodity switches allow configuration of a different marking/drop profile for non-TCP and non-IP packets. Non-TCP and non-IP packets should be able to pass through the switch unless the switch is really out of buffers. If the traffic in the datacenter consists of such traffic (including UDP), one possible mitigation would be to mark IP packets as ECT even when there is no transport that is reacting to the marking.

Since DCTCP relies on congestion marking by the switch, DCTCP can only be deployed in datacenters where the network infrastructure supports ECN. The switches may also support configuration of the congestion threshold used for marking. The proposed parameterization can be configured with switches that implement RED. [DCTCP10] provides a theoretical basis for selecting the congestion threshold, but as with estimation gain, it may be more practical to rely on experimentation or simply to use the default configuration of the device. DCTCP will degrade to loss-based congestion control when transiting a congested drop-tail link.

DCTCP requires changes on both the sender and the receiver, so both endpoints must support DCTCP. Furthermore, DCTCP provides no mechanism for negotiating its use, so both endpoints must be configured through some out-of-band mechanism to use DCTCP. A variant of DCTCP that can be deployed unilaterally and only requires standard ECN behavior has been described in [ODCTCP][BSDCAN], but requires additional experimental evaluation.

6. Known Issues

DCTCP relies on the sender's ability to reconstruct the stream of CE codepoints received by the remote endpoint. To accomplish this, DCTCP avoids using a single ACK packet to acknowledge segments received both with and without the CE codepoint set. However, if one or more ACK packets are dropped, it is possible that a subsequent ACK will cumulatively acknowledge a mix of CE and non-CE segments. This will, of course, result in a less accurate congestion estimate. There are some potential mitigations:

- o Even with a degraded congestion estimate, DCTCP may still perform better than [RFC3168].
- o If the estimation gain is small relative to the packet loss rate, the estimate may not be degraded much.
- o If packet losses mostly occur under heavy congestion, most drops will occur during an unbroken string of CE packets, and the estimate will be unaffected.

However, the affect of packet drops on DCTCP under real world conditions has not been analyzed.

DCTCP provides no mechanism for negotiating its use. Thus, there is additional management and configuration overhead required to ensure that DCTCP is not used with non-DCTCP endpoints. The affect of using DCTCP with a standard ECN endpoint has been analyzed in [ODCTCP][BSDCAN]. Furthermore, it is possible that other

implementations may also modify [RFC3168] behavior without negotiation, causing further interoperability issues.

Much like standard TCP, DCTCP is biased against flows with longer RTTs. A method for improving the fairness of DCTCP has been proposed in [ADCTCP], but requires additional experimental evaluation.

7. Implementation Status

This section documents the implementation status of the specification in this document, as recommended by [RFC6982].

This document describes DCTCP as implemented in Microsoft Windows Server 2012. Since publication of the first versions of this document, the Linux [LINUX] and FreeBSD [FREEBSD] operating systems have also implemented support for DCTCP in a way that is believed to follow this document.

8. Security Considerations

DCTCP enhances ECN and thus inherits the security considerations discussed in [RFC3168]. The processing changes introduced by DCTCP do not exacerbate these considerations or introduce new ones. In particular, with either algorithm, the network infrastructure or the remote endpoint can falsely report congestion and thus cause the sender to reduce cwnd. However, this is no worse than what can be achieved by simply dropping packets.

9. IANA Considerations

This document has no actions for IANA.

10. Acknowledgements

The DCTCP algorithm was originally proposed and analyzed in [DCTCP10] by Mohammad Alizadeh, Albert Greenberg, Dave Maltz, Jitu Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan.

Lars Eggert has received funding from the European Union's Horizon 2020 research and innovation program 2014-2018 under grant agreement No. 644866 ("SSICLOPS"). This document reflects only the authors' views and the European Commission is not responsible for any use that may be made of the information it contains.

11. References

11.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [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.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

11.2. Informative References

- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, June 2009.
- [RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, July 2013.
- [DCTCP10] Alizadeh, M., Greenberg, A., Maltz, D., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and M. Sridharan, "Data Center TCP (DCTCP)", Proc. ACM SIGCOMM 2010 Conference (SIGCOMM 10), August 2010, <<http://dl.acm.org/citation.cfm?doid=1851182.1851192>>.
- [ODCTCP] Kato, M., "Improving Transmission Performance with One-Sided Datacenter TCP", M.S. Thesis, Keio University, 2014, <<http://eggert.org/students/kato-thesis.pdf>>.
- [BSDCAN] Kato, M., Eggert, L., Zimmermann, A., van Meter, R., and H. Tokuda, "Extensions to FreeBSD Datacenter TCP for Incremental Deployment Support", BSDCan 2015, June 2015, <<https://www.bsdcn.org/2015/schedule/events/559.en.html>>.

- [ADCTCP] Alizadeh, M., Javanmard, A., and B. Prabhakar, "Analysis of DCTCP: Stability, Convergence, and Fairness", Proc. ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 11), June 2011, <<https://dl.acm.org/citation.cfm?id=1993753>>.
- [LINUX] Borkmann, D. and F. Westphal, "Linux DCTCP patch", 2014, <<https://git.kernel.org/cgit/linux/kernel/git/davem/net-next.git/commit/?id=e3118e8359bb7c59555aca60c725106e6d78c5ce>>.
- [FREEBSD] Kato, M. and H. Panchasara, "DCTCP (Data Center TCP) implementation", 2015, <<https://github.com/freebsd/freebsd/commit/8ad879445281027858a7fa706d13e458095b595f>>.
- [MORGANSTANLEY] Judd, G., "Attaining the Promise and Avoiding the Pitfalls of TCP in the Datacenter", Proc. 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), May 2015, <<https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/judd>>.

Authors' Addresses

Stephen Bensley
Microsoft
One Microsoft Way
Redmond, WA 98052
USA

Phone: +1 425 703 5570
Email: sbens@microsoft.com

Lars Eggert
NetApp
Sonnenallee 1
Kirchheim 85551
Germany

Phone: +49 151 120 55791
Email: lars@netapp.com
URI: <http://eggert.org/>

Dave Thaler
Microsoft

Phone: +1 425 703 8835
Email: dthaler@microsoft.com

Praveen Balasubramanian
Microsoft

Phone: +1 425 538 2782
Email: pravb@microsoft.com

Glenn Judd
Morgan Stanley

Phone: +1 973 979 6481
Email: glenn.judd@morganstanley.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: August 18, 2014

A. Bittau
D. Boneh
M. Hamburg
Stanford University
M. Handley
University College London
D. Mazieres
Q. Slack
Stanford University
February 14, 2014

Cryptographic protection of TCP Streams (tcpcrypt)
draft-bittau-tcp-crypt-04.txt

Abstract

This document presents tcpcrypt, a TCP extension for cryptographically protecting TCP segments. Tcpcrypt maintains the confidentiality of data transmitted in TCP segments against a passive eavesdropper. It protects connections against denial-of-service attacks involving desynchronizing of sequence numbers, and when enabled, against forged RST segments. Finally, applications that perform authentication can obtain end-to-end confidentiality and integrity guarantees by tying authentication to tcpcrypt Session ID values.

The extension defines two new TCP options, CRYPT and MAC, which are designed to provide compatible interworking with TCPs that do not implement tcpcrypt. The CRYPT option allows hosts to negotiate the use of tcpcrypt and establish shared secret encryption keys. The MAC option carries a message authentication code with which hosts can verify the integrity of transmitted TCP segments. Tcpcrypt is designed to require relatively low overhead, particularly at servers, so as to be useful even in the case of servers accepting many TCP connections per second.

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 18, 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Requirements Language	5
2. Introduction	5
3. Idealized protocol	5
3.1. Stages of the protocol	5
3.1.1. The setup phase	6
3.1.2. The ENCRYPTING state	6
3.1.3. The DISABLED state	7
3.2. Cryptographic algorithms	7
3.3. "C" and "S" roles	9
3.4. Key exchange protocol	9
3.5. Data encryption and authentication	11
3.6. Authenticated Sequence Mode (ASM)	12
3.6.1. ASM-Encrypt	14
3.6.2. ASM-Decrypt	15
3.6.3. ASM-Update	15
3.7. Re-keying	16
3.8. Session caching	16
3.8.1. Session caching control	17
4. Extensions to TCP	17
4.1. Protocol states	18
4.2. Role negotiation	22
4.2.1. Simultaneous open	23
4.3. The TCP CRYPT option	24
4.3.1. The HELLO suboption	27
4.3.2. The DECLINE suboption	28
4.3.3. The NEXTK1 and NEXTK2 suboptions	28
4.3.4. The PKCONF suboption	30
4.3.5. The UNKNOWN suboption	31
4.3.6. The SYNCOOKIE and ACKCOOKIE suboptions	31
4.3.7. The SYNC_REQ and SYNC_OK suboptions	32
4.3.8. The REKEY and REKEYSTREAM suboptions	34
4.3.9. The INIT1 and INIT2 suboptions	36
4.3.10. The IV suboption	38
4.4. The TCP MAC option	39
5. Examples	41
5.1. Example 1: Normal handshake	41
5.2. Example 2: Normal handshake with SYN cookie	41
5.3. Example 3: tcpcrypt unsupported	42
5.4. Example 4: Reusing established state	42
5.5. Example 5: Decline of state reuse	42
5.6. Example 6: Reversal of client and server roles	42
6. API extensions	43
7. Acknowledgments	45
8. IANA Considerations	45
9. Security Considerations	48
10. References	48

10.1. Normative References	48
10.2. Informative References	49
Appendix A. Protocol constant values	50
Authors' Addresses	50

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. Introduction

This document describes tcpcrypt, an extension to TCP for cryptographic protection of session data. Tcpcrypt was designed to meet the following goals:

- o Maintain confidentiality of communications against a passive adversary. Ensure that an adversary must actively intercept and modify the traffic to eavesdrop, either by re-encrypting all traffic or by forcing a downgrade to an unencrypted session.
- o Minimize computational cost, particularly on servers.
- o Provide interfaces to higher-level software to facilitate end-to-end security, either in the application level protocol or after the fact. (E.g., client and server log session IDs and can compare them after the fact; if there was no tampering or eavesdropping, the IDs will match.)
- o Be compatible with further extensions that allow authenticated resumption of TCP connections when either end changes IP address.
- o Facilitate multipath TCP [RFC6824] by identifying a TCP stream with a session ID independent of IP addresses and port numbers.
- o Provide for incremental deployment and graceful fallback, even in the presence of NATs and other middleboxes that might remove unknown options, and traffic normalizers.

3. Idealized protocol

This section describes the tcpcrypt protocol at an abstract level, without reference to particular cryptographic algorithms or data encodings. Readers who simply wish to see the key exchange protocol should skip to Section 3.4.

3.1. Stages of the protocol

A tcpcrypt endpoint goes through multiple stages. It begins in a setup phase and ends up in one of two states, ENCRYPTING or DISABLED,

before applications may send or receive data. The ENCRYPTING and DISABLED states are definitive and mutually exclusive; an endpoint that has been in one of the two states MUST NOT ever enter the other, nor ever re-enter the setup phase.

3.1.1. The setup phase

The setup phase negotiates use of the tcpcrypt extension. During this phase, two hosts agree on a suite of cryptographic algorithms and establish shared secret session keys.

The setup phase uses the Data portion of TCP segments to exchange cryptographic keys. Implementations MUST NOT include application data in TCP segments during setup and MUST NOT allow applications to read or write data. System calls MUST behave the same as for TCP connections that have not yet entered the ESTABLISHED state; calls to read and write SHOULD block or return temporary errors, while calls to poll or select SHOULD consider connections not ready.

When setup succeeds, tcpcrypt enters the ENCRYPTING state. Importantly, a successful setup also produces an important value called the `_Session ID_`. The Session ID is tied to the negotiated algorithms and cryptographic keys, and is unique over all time with overwhelming probability.

Operating systems MUST make the Session ID available to applications. To prevent man-in-the-middle attacks, applications MAY authenticate the session ID through any protocol that ensures both endpoints of a connection have the same value. Applications MAY alternatively just log Session IDs so as to enable attack detection after the fact through comparison of the values logged at both ends.

The setup phase can also fail for various reasons, in which case tcpcrypt enters the DISABLED state.

Applications MAY test whether setup succeeded by querying the operating system for the Session ID. Requests for the Session ID MUST return an error when tcpcrypt is not in the ENCRYPTING state. Applications SHOULD authenticate the returned Session ID. Applications relying on tcpcrypt for security SHOULD authenticate the Session ID and SHOULD treat unauthenticated Session IDs the same as connections in the DISABLED state.

3.1.2. The ENCRYPTING state

When the setup phase succeeds, tcpcrypt enters the ENCRYPTING state. Once in this state, applications may read and write data with the expected semantics of TCP connections.

In the ENCRYPTING state, a host MUST encrypt the Data portion of all TCP segments transmitted and MUST include a Message Authentication Code (MAC) in all segments transmitted. A host MUST furthermore ignore any TCP segments received without the RST bit set, unless those segments also contain a valid MAC option.

A host SHOULD accept RST segments without valid MACs by default. However, the application SHOULD be allowed to force unMACed RST segments to be dropped by enabling the TCP_CRYPT_RSTCHK option on the connection.

Once in the ENCRYPTING state, an endpoint MUST NOT directly or indirectly transition to the DISABLED state under any circumstances.

3.1.3. The DISABLED state

When setup fails, tcpcrypt enters the DISABLED state. In this case, the host MUST continue just as TCP would without tcpcrypt, unless network conditions would cause a plain TCP connection to fail as well. Entering the DISABLED state prohibits the endpoint from ever entering the ENCRYPTING state.

An implementation MUST behave identically to ordinary TCP in the DISABLED state, except that the first segment transmitted after entering the DISABLED state MAY include a TCP CRYPT option with a DECLINE suboption (and optionally other suboptions such as UNKNOWN) to indicate that tcpcrypt is supported but not enabled. Section 4.3.2 describes how this is done.

Operating systems MUST allow applications to turn off tcpcrypt by setting the state to DISABLED before opening a connection. An active opener with tcpcrypt disabled MUST behave identically to an implementation of TCP without tcpcrypt. A passive opener with tcpcrypt disabled MUST also behave like normal TCP, except that it MAY optionally respond to SYN segments containing a CRYPT option with SYN-ACK segments containing a DECLINE suboption, so as to indicate that tcpcrypt is supported but not enabled.

3.2. Cryptographic algorithms

The setup phase employs three types of cryptographic algorithms:

- o A _public key cipher_ is used with a short-lived public key to exchange (or agree upon) a random, shared secret.
- o An _extract function_ is used to generate a pseudo-random key from some initial keying material, typically the output of the public key cipher. The notation Extract (S, IKM) denotes the output of

the extract function with salt *S* and initial keying material *IKM*.

- o A *_collision-resistant pseudo-random function (CPRF)_* is used to generate multiple cryptographic keys from a pseudo-random key, typically the output of the extract function. We use the notation CPRF (*K*, *TAG*, *L*) to designate the output of *L* bytes of the pseudo-random function identified by key *K* on *TAG*. A collision-resistant function is one on which, for sufficiently large *L*, an attacker cannot find two distinct inputs *K*₁, *TAG*₁ and *K*₂, *TAG*₂ such that CPRF (*K*₁, *TAG*₁, *L*) = CPRF (*K*₂, *TAG*₂, *L*). Collision resistance is important to assure the uniqueness of Session IDs, which are generated using the CPRF.

The Extract and CPRF functions used by default are the Extract and Expand functions of HKDF [RFC5869]. These are defined as follows:

```

HKDF-Extract(salt, IKM) -> PRK
    PRK = HMAC-Hash(salt, IKM)

HKDF-Expand(PRK, TAG, L) -> OKM
    T(0) = empty string (zero length)
    T(1) = HMAC-Hash(PRK, T(0) | TAG | 0x01)
    T(2) = HMAC-Hash(PRK, T(1) | TAG | 0x02)
    T(3) = HMAC-Hash(PRK, T(2) | TAG | 0x03)
    ...

    OKM = first L octets of T(1) | T(2) | T(3) | ...

```

The symbol *|* denotes concatenation, and the counter concatenated with *TAG* is a single octet.

Because the public key cipher, the extract function, and the expand function all make use of cryptographic hashes in their constructions, the three algorithms are negotiated as a unit employing a single hash function. For example, the OAEP+-RSA [RFC2437] cipher, which uses a SHA-256-based mask-generation function, is coupled with HKDF [RFC5869] using HMAC-SHA256 [RFC2104].

The encrypting phase employs an *_authenticated encryption mode_* to encrypt all application data. This mode authenticates both application data and most of the TCP header (excepting header fields commonly modified by middleboxes).

Note that public key generation, public key encryption, and shared secret generation all require randomness. Other tcpcrypt functions may also require randomness depending on the algorithms and modes of operation selected. A weak pseudo-random generator at either host will defeat tcpcrypt's security. Thus, any host implementing

tcpcrypt MUST have a cryptographically secure source of randomness or pseudo-randomness.

3.3. "C" and "S" roles

Tcpcrypt transforms a single pseudo-random key (PRK) into cryptographic session keys for each direction. Doing so requires an asymmetry in the protocol, as the key derivation function must be perturbed differently to generate different keys in each direction. Tcpcrypt includes other asymmetries in the roles of the two hosts, such as the process of negotiating algorithms (e.g., proposing vs. selecting cipher suites).

We use the terms "C" and "S" to denote the distinct roles of the two hosts in tcpcrypt's setup phase. In the case of key transport, "C" is the host that supplies a public key, while "S" is the host that encrypts a pre-master secret with the key belonging to "C". Which role a host plays can have performance implications, because for some public key algorithms encryption is much faster than decryption. For instance, on a machine at the time of writing, encryption with a 2,048-bit RSA-3 key costs 82 microseconds, while decryption costs 10 milliseconds.

Because servers often need to establish connections at a faster rate than clients, and because servers are often passive openers, by default the passive opener plays the "S" role. However, operating systems MUST provide a mechanism for the passive opener to reverse roles and play the "C" role, as discussed in Section 4.2.

3.4. Key exchange protocol

Every machine C has a short-lived public encryption key or key agreement parameter, PK_C, which gets refreshed periodically and SHOULD NOT ever be written to persistent storage.

When a host C connects to S, the two engage in the following protocol:

```
C -> S: HELLO
S -> C: PKCONF, pub-cipher-list
C -> S: INIT1, sym-cipher-list, N_C, PK_C
S -> C: INIT2, sym-cipher, KX_S
```

The parameters are defined as follows:

- o pub-cipher-list: a list of public key ciphers and parameters acceptable to S. These are defined in Figure 3.

- o sym-cipher-list: a list of symmetric cipher suites acceptable to C. These are specified in Table 6.
- o N_C: Nonce chosen at random by C.
- o PK_C: C's public key or key agreement parameter.
- o sym-cipher: the symmetric cipher selected by S.
- o KX_S: key exchange information produced by S. KX_S will depend on whether key transport is being done (e.g., RSA) or key agreement (e.g., Diffie-Hellman). KX_S is defined in Table 1.

Cipher	KX_S	PMS
OAEP+-RSA exp3	ENC (PK_C, R_S)	R_S
ECDHE	N_S, PK_S	key-agreement-output

ENC (PK_C, R_S) denotes an encryption of R_S with public key PK_C. R_S and N_S are random values chosen by S. Their lengths are defined in Figure 3. PK_S is S's key agreement parameter. PMS is the Pre Master Secret from which keys are ultimately derived.

Table 1

The two sides then compute a pseudo-random key (PRK) from which all session keys are derived as follows:

```
param    := { pub-cipher-list, sym-cipher-list, sym-cipher }
PRK      := Extract (N_C, { param, PK_C, KX_S, PMS })
```

A series of "session secrets" and corresponding Session IDs are then computed as follows:

```
ss[0] := PRK
ss[i] := CPRF (ss[i-1], CONST_NEXTK, K_LEN)

SID[i] := CPRF (ss[i], CONST_SESSID, K_LEN)
```

The value ss[0] is used to generate all key material for the current connection. SID[0] is the session ID for the current connection, and will with overwhelming probability be unique for each individual TCP connection. The most computationally expensive part of the key exchange protocol is the public key cipher. The values of ss[i] for $i > 0$ can be used to avoid public key cryptography when establishing subsequent connections between the same two hosts, as described in

Section 3.8. The TAG values are constants defined in Table 7. The K_LEN values along with nonce sizes are negotiated, and are specified in Figure 3.

Given a session secret, *ss*, the two sides compute a series of master keys as follows:

```
mk[0] := CPRF (ss, CONST_REKEY, K_LEN)
mk[i] := CPRF (mk[i-1], CONST_REKEY, K_LEN)
```

Finally, each master key *mk* is used to generate keys for authenticated encryption for the "S" and "C" roles. Key *k_{cs}* is used by "C" to encrypt and "S" to decrypt, while *k_{sc}* is used by "S" to encrypt and "C" to decrypt.

```
k_cs := CPRF (mk, CONST_KEY_C, ae_len)
k_sc := CPRF (mk, CONST_KEY_S, ae_len)
```

tcpcrypt does not use HKDF directly for key derivation because it requires multiple expand steps with different keys. This is needed for forward secrecy so that *ss*[*n*] can be forgotten once a session is established, and *mk*[*n*] can be forgotten once a session is rekeyed.

There is no key confirmation step in tcpcrypt. This is not required since in tcpcrypt's threat model, a connection to an adversary can be made and so keys need not be verified. If an erroneous key negotiation that yields two different keys occurs, all subsequent packets will be dropped due to an incorrect MAC, causing the TCP connection to hang. This is not a threat because in plain TCP, an active attacker could have modified sequence and ack numbers to hang the connection anyway.

3.5. Data encryption and authentication

tcpcrypt encrypts and authenticates all application data. It also authenticates some parts of the TCP header. There are several TCP-specific constraints with regards to authenticated encryption that tcpcrypt must meet for performance and compatibility with middleboxes:

- o The ciphertext for a particular byte position in tcpcrypt's sequence must never change, even if reencryption occurs after coalescing and retransmission. This is because a middlebox may discard a changed payload on retransmission.
- o Authentication must occur only on fields not modified by middleboxes. In particular, port numbers must not be authenticated, and sequence and ack numbers must be authenticated

according to an offset from the initial sequence number, because these can be modulated by a middlebox.

- o An efficient mechanism is needed for recomputing the authentication tag when only the ack numbers change. For example, on retransmissions, the authenticated encryption authentication tag can be efficiently updated without having to recompute the tag on the entire packet payload.

Authenticated encryption modes such as GCM do not meet these criteria. For example, even with identical plaintext, ciphertext values depend on the byte position at which one starts encrypting a segment. Hence two small segments will appear to have different content from their coalesced counterpart; middleboxes might drop such coalesced retransmissions after falsely detecting subterfuge attacks. Furthermore, existing authenticated encryption modes do not allow efficient updating of the authentication tag when only small parts of the data have changed. A new mode is needed to meet all these constraints, and we introduce `_Authenticated Sequence Mode_ (ASM)` in Section 3.6 as a solution.

ASM takes three parameters: a cipher, a MAC and an ACK MAC. At a high-level, the cipher is used to encrypt the TCP payload in counter mode, using a counter derived from TCP's sequence number. The MAC covers the ciphertext and parts of the TCP header. The ACK MAC covers the ACK numbers and is XORed with the previously computed MAC to produce the authenticated encryption authentication tag. This tag can be quickly updated if only the ACK numbers have changed. This approach is principled because ACK messages are conceptually separate from data packets, so MACing them separately is appropriate. In TCP, ACKs are piggybacked to data segments merely as an optimization.

XORing two PRF-based MACs together was shown secure by Katz and Lindell [aggregate-macs].

3.6. Authenticated Sequence Mode (ASM)

ASM is parameterized by a cipher, MAC and ACK MAC. The operations supported by ASM are:

```
ASM-Encrypt (PRK, Seq, Message, Assoc-Data, Up-Data) ->
              (Ciphertext, Auth-Tag)
```

```
ASM-Decrypt (PRK, Seq, Cipher-Text, Assoc-Data, Up-Data, Auth-Tag) ->
              { (Valid, Message)  OR
                (Invalid, )
              }
```

```
ASM-Update (PRK, Up-Data-Prev, Up-Data-New, Auth-Tag-Prev) ->
              Auth-Tag
```

The arguments and return values are:

- o `_PRK_` a pseudo-random key.
- o `_Seq_` the byte position in the stream of Message or Cipher-Text. In tcpcrypt, this is an extended version of TCP's sequence number.
- o `_Message_` the Message to encrypt. In tcpcrypt, this is TCP's payload.
- o `_Assoc-Data_` the associated data to be MACed but not encrypted. In tcpcrypt, this contains parts of the TCP header.
- o `_Up-Data_` the updatable data to be MACed but not encrypted, that can also be efficiently updated and reMACed. In tcpcrypt, this will cover an extended version of TCP's ACK numbers.
- o `_Ciphertext_` the encrypted version of Message.
- o `_Auth-Tag_` the authenticated encryption authentication tag. In tcpcrypt, this will be the MAC option.

ASM-Decrypt either returns the Valid or Invalid constants, depending on whether the authentication tag can be verified successfully or not. For Valid inputs, the Message is returned as well.

The PRK supplied to ASM is expanded into keys used for individual operation as follows:

```
k_enc := CPRF (PRK, CONST_KEY_ENC, cipher-key-len)
k_mac := CPRF (PRK, CONST_KEY_MAC, mac-key-len)
k_ack := CPRF (PRK, CONST_KEY_ACK, ack-mac-key-len)
```

The next sections describe ASM operations in detail.

3.6.1. ASM-Encrypt

The interface to encrypt is as follows:

```
ASM-Encrypt (PRK, Seq, Message, Assoc-Data, Up-Data) ->
              (Ciphertext, Auth-Tag)
```

Keys (denoted by k_*) are derived from PRK as explained in Section 3.6.

The following steps occur:

1. Message is encrypted to produce Ciphertext using the cipher in counter mode. Seq is the counter and k_{enc} is the key. When encrypting Seq, its value must always be a multiple of the cipher's block size. In the event that the message does not begin on an even block boundary, Seq must be rounded down, encrypted, and leading bytes of its encryption discarded.
2. The MAC is run over the concatenation of Ciphertext and Assoc-Data to produce MAC1, using k_{mac} as the key.
3. The ACK MAC is run over Up-Data to produce MAC2, using k_{ack} as the key.
4. MAC1 and MAC2 are XORed to produce Auth-Tag.

Using AES-128 as an example, encryption in counter mode using Seq as the counter happens as follows.

- o Compute $B = Seq - (Seq \% 16)$.
- o Let $B^* = 0^{128-|B|}$ | B be B in network (big-endian) byte order with enough 0 bits pre-pended to make B^* exactly 128 bits long.
- o Let $C = ENC\text{-}AES(k_e, B^*)$.
- o Discard the first (Seq-B) bytes on C and begin byte-by-byte XORing the remaining portion with the message.

If AES-128 is used as the ACK MAC, the Ack number (64-bit extended, offset from ISN) is first padded on the left with enough zeros to produce a 128-bit big-endian value. The number is then encrypted using AES.

3.6.2. ASM-Decrypt

The interface to decrypt is as follows:

```
ASM-Decrypt (PRK, Seq, Cipher-Text, Assoc-Data, Up-Data, Auth-Tag) ->
    { (Valid, Message) OR
      (Invalid, )
```

Keys (denoted by k_*) are derived from PRK as explained in Section 3.6.

The following steps occur:

1. The MAC is run over the concatenation of Ciphertext and Assoc-Data to produce MAC1, using k_mac as the key.
2. The ACK MAC is run over Up-Data to produce MAC2, using k_ack as the key.
3. MAC1 and MAC2 are XORed and compared to Auth-Tag. If different, the process stops and the constant Invalid is returned along with no message. Otherwise the process continues.
4. Ciphertext is decrypted to produce Message using the cipher in counter mode. Seq is the counter and k_enc is the key. The Valid constant is returned along with Message.

3.6.3. ASM-Update

The interface to update the authenticated encryption authentication tag is as follows:

```
ASM-Update (PRK, Up-Data-Prev, Up-Data-New, Auth-Tag-Prev) ->
    Auth-Tag
```

Keys (denoted by k_*) are derived from PRK as explained in Section 3.6.

The following steps occur:

1. The ACK MAC is run over Up-Data-Prev using k_ack to produce MAC2-Prev.
2. MAC2-Prev is XORed with Auth-Tag-Prev to produce MAC1.
3. The ACK MAC is run over Up-Data to produce MAC2, using k_ack as the key.

4. MAC1 and MAC2 are XORed to produce Auth-Tag.

3.7. Re-keying

We refer to the two encryption keys (k_{cs} , k_{sc}) as a key set. We refer to the key set generated by $mk[i]$ as the key set with generation number i within a session. Initially, the two hosts use the key set with generation number 0.

Either host may decide to evolve the encryption key at one or more points within a session, by incrementing the generation number of its transmit keys. When switching keys to generation j , a host must label the segments it transmits with a REKEY option containing j , so that the recipient host knows to check the MAC and decrypt the segment using the new keyset:

A -> B: REKEY< j >, MAC<...>, Data<...>

Upon receiving a REKEY< j > segment, a recipient using transmit keys from a generation less than j must also update its transmit keys and start including a REKEY< j > option in all of its segments. A host must continue transmitting REKEY options until all segments with other generation numbers have been processed at both ends.

Implementations **MUST** always transmit and retransmit identical ciphertext Data bytes for the same TCP sequence numbers. Thus, a retransmitted segment **MUST** always use the same keyset as the original segment. Hosts **MUST NOT** combine segments that were encrypted with different keysets.

Implementations **SHOULD** delete older-generation keys from memory once they have received all segments they will need to decrypt with the old keys and received acknowledgments for all segments they might need to retransmit.

3.8. Session caching

When two hosts have already negotiated session secret $ss[i-1]$, they can establish a new connection without public key operations using $ss[i]$. The four-message protocol of Section 3.4 is replaced by:

A -> B: NEXTK1, SID[i]
B -> A: NEXTK2

Which symmetric keys a host uses for transmitted segments is determined by its role in the original session $ss[0]$. It does not depend on which host is the passive opener in the current session. If A had the "C" role in the first session, then A uses k_{cs} for

sending segments and `k_sc` for receiving. Otherwise, if A had the "S" role originally, it uses `k_sc` and `k_cs`, respectively. B similarly uses the transmit keys that correspond to its role in the original session.

After using `ss[i]` to compute `mk[0]`, implementations SHOULD compute and cache `ss[i+1]` for possible use by a later session, then erase `ss[i]` from memory. Hosts SHOULD keep `ss[i+1]` around for a period of time until it is used or the memory needs to be reclaimed. Hosts SHOULD NOT write a cached `ss[i+1]` value to non-volatile storage.

It is an implementation-specific issue as to how long `ss[i+1]` should be retained if it is unused. If the passive opener times it out before the active opener does, the only cost is the additional twelve bytes to send NEXTK1 for the next connection. The behavior then falls back to a normal public-key handshake.

3.8.1. Session caching control

Implementations MUST allow applications to control session caching by setting the following option:

`TCP_CRYPT_CACHE_FLUSH` When set on a TCP endpoint that is in the ENCRYPTING state, this option causes the operating system to flush from memory the cached `ss[i+1]` (or `ss[i+1+n]` if other connections have already been established). When set on an endpoint that is in the setup phase, causes any cached `ss[i]` that would have been used to be flushed from memory. In either case, future connections will have to undertake another round of the public key protocol in Section 3.4. Applications SHOULD set `TCP_CRYPT_CACHE_FLUSH` whenever authentication of the session ID fails.

4. Extensions to TCP

The `tcpcrypt` extension adds two new kinds of option: `CRYPT`, and `MAC`. Both are described in this section. During the setup phase, all TCP segments MUST have the `CRYPT` option. In the ENCRYPTING state, all segments MUST have the `MAC` option and may include the `CRYPT` option for various purposes such as re-keying or keep-alive probes.

The idealized protocol of the previous section must be embedded in the TCP handshake. Unfortunately, since the maximum TCP header size is 60 bytes and the basic TCP header fields require 20 bytes, there are at most 40 option payload bytes available, which is not enough to hold the INIT1 and INIT2 messages. `Tcp`crypt therefore uses the Data portion of TCP segments (after the SYN exchanges) to send the body of

these messages.

Operating systems MUST keep track of which phase a data segment belongs to, and MUST only deliver data to applications from segments that are processed in the ENCRYPTING or DISABLED states.

4.1. Protocol states

The setup phase is divided into six states: CLOSED, NEXTK-SENT, HELLO-SENT, C-MODE, LISTEN, and S-MODE. Together with the ENCRYPTING and DISABLED states already discussed, this means a tcpcrypt endpoint can be in one of eight states.

In addition to tcpcrypt's state, each endpoint will also be in one of the 11 TCP states described in the TCP protocol specification [RFC0793]. Not all pairs of states are valid. Table 2 shows which TCP states an endpoint can be in for each tcpcrypt state.

Tcpcrypt state	TCP states for an active opener	TCP states for a passive opener
CLOSED	CLOSED	CLOSED
NEXTK-SENT	SYN-SENT	n/a
HELLO-SENT	SYN-SENT	SYN-RCVD
C-MODE	ESTABLISHED, FIN-WAIT-1	ESTABLISHED, FIN-WAIT-1
LISTEN	n/a	LISTEN
S-MODE	(SYN-RCVD), ESTABLISHED	SYN-RCVD
ENCRYPTING	(SYN-RCVD), ESTABLISHED+	SYN-RCVD, ESTABLISHED+
DISABLED	any	any

Valid tcpcrypt and TCP state combinations. States in parentheses occur only with simultaneous open. ESTABLISHED+ means ESTABLISHED or any later state (FIN-WAIT-1, FIN-WAIT-2, CLOSING, TIME-WAIT, CLOSE-WAIT, or LAST-ACK).

Table 2

Figure 1 shows how tcpcrypt transitions between states. Each transition is labeled by events that may trigger the transition above the line, and an action the local host is permitted to take in response below the line. "snd" and "rcv" denote sending and receiving segments, respectively. "any" means any possible event. "internal" means any possible event except for receiving a segment (i.e., timers and system calls). "drop" means discarding the last received segment and preventing it from having any effect on TCP's state. "mac" means any valid TCP action, including no action, except that any segments

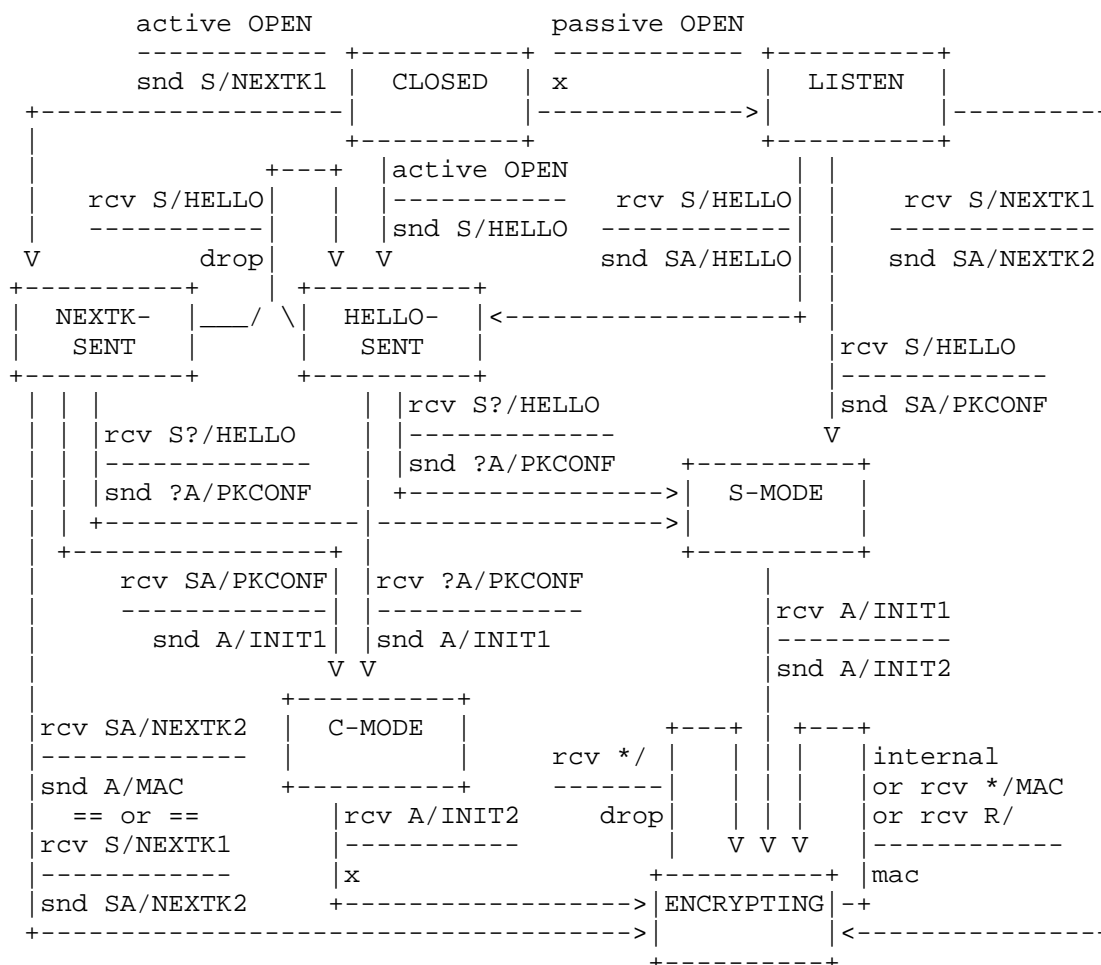
transmitted must be encrypted and contain a valid TCP MAC option. "x" indicates that a host sends no segments when taking a transition.

A segment is described as "F/Op". F specifies constraints on the control bits of the TCP header, as follows:

F	Meaning
S	SYN=1, ACK=0, FIN=0, RST=0
SA	SYN=1, ACK=1, FIN=0, RST=0
A	SYN=0, ACK=1, FIN=0, RST=0
S?	SYN=1, ACK=any, FIN=0, RST=0
?A	SYN=any, ACK=1, FIN=0, RST=0
R	RST=1
*	any

Op designates message types in the abstract protocol, which also correspond to particular suboptions of the TCP CRYPT option, described in Section 4.3, or "MAC" for a valid TCP MAC option, as described in Section 4.4. A segment with SYN=1 and ACK=0 that contains the NEXTK1 suboption will also explicitly or implicitly contain the HELLO suboption; such a segment matches event constraints on either option--e.g., it matches any of the "rcv S/HELLO", "rcv S?/HELLO", and "rcv S/NEXTK1" events. An empty Op matches any segment with the appropriate control bits. A segment MUST contain the TCP MAC option if and only if Op is "MAC".

The "drop" transitions from NEXTK-SENT and HELLO-SENT to HELLO-SENT change TCP slightly by ignoring a segment and preventing a TCP transition from SYN-SENT to SYN-RCVD that would otherwise occur during simultaneous open. Therefore, these transitions SHOULD be disabled by default. They MAY be enabled on one side by an application that wishes to enable tcpcrypt on simultaneous open, as discussed in Section 4.2.1.



State diagram for tcpcrypt. Transitions to DISABLED and CLOSED are not shown.

Figure 1

Any segment that would be discarded by TCP (e.g., for being out of window) MUST also be ignored by tcpcrypt. However, certain segments that might otherwise be accepted by TCP MUST be dropped by tcpcrypt and prevented from affecting TCP's state.

Except for these drop actions, tcpcrypt MUST abide by the TCP protocol specification [RFC0793]. Thus, any segment transmitted by a host MUST be permitted by the TCP specification in addition to matching either a transition in Figure 1 or one of the transitions to

DISABLED or CLOSED described below. In particular, a host MUST NOT acknowledge an INIT1 segment unless either the acknowledgment contains an INIT2 or the host transitions to DISABLED.

Various events cause transitions to DISABLED from states other than ENCRYPTING. In particular:

- o Operating systems MUST provide a mechanism for applications to transition to DISABLED from the CLOSED and LISTEN states.
- o A host in the setup phase MUST transition to DISABLED upon receiving any segment without a TCP CRYPT option.
- o A host in the setup phase MUST transition to DISABLED upon receiving any segment with the FIN or RST control bit set.
- o A host in the setup phase MUST transition to DISABLED upon sending a segment with the FIN bit set. (As discussed below, however, a host MUST NOT send a FIN segment from the C-MODE state.)

Other specific conditions cause a transition to DISABLED and are discussed in the sections that follow.

CLOSED is a pseudo-state representing a connection that does not exist. A tcpcrypt connection's lifetime is identical to that of its associated TCP connection. Thus, tcpcrypt transitions to CLOSED exactly when TCP transitions to CLOSED.

A host MUST NOT send a FIN segment from the C-MODE state. The reason is that the remote side can be in the ENCRYPTING state and would thus require the segment to contain a valid MAC, yet a host in C-MODE cannot compute the necessary encryption keys before receiving the INIT2 segment.

If a CLOSE happens in C-MODE, a host MUST delay sending a FIN segment until receiving an ACK for its INIT1 segment. If the remote host is in ENCRYPTING, the ACK segment will contain INIT2 and the local host can transition to ENCRYPTING before sending the FIN. If the remote host is not in ENCRYPTING, the ACK will not contain INIT2, and thus the local host can transition to DISABLED before sending the FIN.

If a CLOSE happens in C-MODE, an implementation MAY delay processing the CLOSE event and entering the TCP FIN-WAIT-1 state until sending the FIN. If it does not, the implementation MUST ensure all relevant timers correspond to the time of transmission of the FIN segment, not the time of entry into the FIN-WAIT-1 state.

The only valid tcpcrypt state transition from ENCRYPTING is to

CLOSED, which occurs only when TCP transitions to CLOSED. tcpcrypt per-se cannot cause TCP to transition to CLOSED.

4.2. Role negotiation

A passive opener receiving an S/HELLO segment may choose to play the "S" role (by transitioning to S-MODE) or the "C" role (by transitioning to HELLO-SENT). An active opener may accept the role not chosen by the passive opener, or may instead disable tcpcrypt. During simultaneous open, one endpoint must choose the "C" role while the other chooses the "S" role. Operating systems MUST allow applications to guide these choices on a per-connection basis.

Applications SHOULD be able to exert this control by setting a per-connection `_CMODE disposition_`, which can take on one of the following five values:

`TCP_CRYPT_CMODE_DEFAULT` This disposition SHOULD be the default. A passive opener will only play the "S" role, but an active opener can play either the "C" or the "S" role. Simultaneous open without session caching will cause tcpcrypt to be disabled unless the remote host has set the `TCP_CMODE_ALWAYS[_NK]` disposition.

`TCP_CRYPT_CMODE_ALWAYS`

`TCP_CRYPT_CMODE_ALWAYS_NK` With this disposition, a host will only play the "C" role. The `_NK` version additionally prevents the use of session caching if the session was originally established in the "S" role.

`TCP_CRYPT_CMODE_NEVER`

`TCP_CRYPT_CMODE_NEVER_NK` With this disposition, a host will only play the "S" role. The `_NK` version additionally prevents the use of session caching if the session was originally established in the "C" role.

The CMODE disposition prohibits certain state transitions, as summarized in Table 3. If an event occurs for which all valid transitions in Figure 1 are prohibited, a host MUST transition to DISABLED. Operating systems MAY add additional CMODE dispositions, for instance to force or prohibit session caching.

CMODE disposition	Prohibited transitions
TCP_CRYPT_CMODE_DEFAULT	LISTEN --> HELLO-SENT HELLO-SENT --> HELLO-SENT NEXTK-SENT --> HELLO-SENT
TCP_CRYPT_CMODE_ALWAYS[_NK]	any --> S-MODE
TCP_CRYPT_CMODE_NEVER[_NK]	LISTEN --> HELLO-SENT HELLO-SENT --> HELLO-SENT NEXTK-SENT --> HELLO-SENT any --> C-MODE

State transitions prohibited by each CMODE disposition

Table 3

4.2.1. Simultaneous open

During simultaneous open, two ends of a TCP connection are both active openers. If both hosts attempt to use session caching by simultaneously transmitting S/NEXTK1 segments, and if both transmit the same session ID, then both may reply with SA/NEXTK2 segments and immediately enter the ENCRYPTING state. In this case, the host that played "C" when the session was initially negotiated MUST use the symmetric encryption keys for "C" (i.e., encrypt with k_cs, decrypt with k_sc), while the host that initially played "S" uses the "S" keys for the new connection.

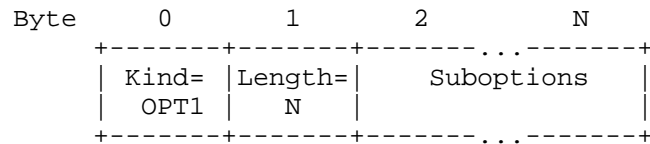
If both hosts in a simultaneous open do not attempt to use session caching, or if the two hosts use incompatible Session IDs, then they MUST engage in public-key-based key negotiation to use tcpcrypt. Doing so requires one host to play the "C" role and the other to play the "S" role. With the TCP_CRYPT_CMODE_DEFAULT disposition, these roles are usually determined by the passive opener choosing the "S" role. With no passive opener, both active openers will end up in S-MODE, then transition to DISABLED upon receiving an unexpected PKCONF.

Simultaneous open can work with key negotiation if exactly one of the two hosts selects the TCP_CRYPT_CMODE_ALWAYS disposition. This host will then drop S/HELLO segments and remain in C-MODE while the other host transitions to S-MODE. Applications SHOULD NOT set TCP_CRYPT_CMODE_ALWAYS on both sides of a simultaneous open, as this will result in tcpcrypt being disabled. The reception of two simultaneous HELLO (or NEXTK) messages will disable tcpcrypt because

it is not explicit as to who is playing the "C" or "S" role.

4.3. The TCP CRYPT option

A CRYPT option has the following format:

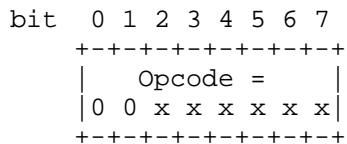


Format of TCP CRYPT option

Kind is always OPT1. Length is the total length of the option, including the two bytes used for Kind and Length. These first two bytes are then followed by zero or more suboptions. Suboptions determine the meaning of the TCP CRYPT option. When a TCP header contains more than one CRYPT option, a host MUST interpret them the same as if all the suboptions appeared in a single CRYPT option. This makes tcpcrypt options future-proof as new suboptions can be placed in a separate CRYPT option, which can be ignored if not understood, while other CRYPT options can still be processed.

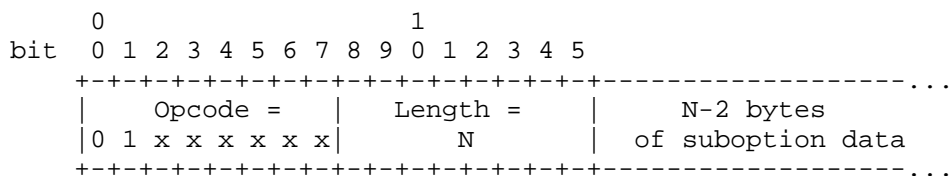
Each suboption begins with an Opcode byte. The specific format of the option depends on the two most significant bits of the Opcode.

Suboptions with opcodes from 0x00 to 0x3f contain no data other than the single opcode byte:



Hosts MUST ignore any opcodes of this format that they do not recognize.

Suboptions with opcodes from 0x40 to 0x7f contain an opcode, a length field, and data bytes.



Hosts MUST ignore any opcodes of this format that they do not recognize.

Suboptions with opcodes from 0x80 to 0xbf contain zero or more bytes of data whose length depends on the opcode. These suboptions can be either fixed length or variable length; implementations that understand these opcodes will know which they are; if the suboption is fixed length the implementation will know the length; otherwise it will know where to look for the length field.

```

bit  0 1 2 3 4 5 6 7
    +---+---+---+---+---+---+---+---+...
    |           Opcode =         | data
    | 1 0 x x x x x x |
    +---+---+---+---+---+---+---+---+...

```

If a host sees an unknown opcode in this range, it MUST ignore the suboption and all subsequent suboptions in the same TCP CRYPT option. However, if more than one CRYPT option appears in the TCP header, the host MUST continue processing suboptions from the next TCP CRYPT option. Skipping suboptions in the TCP CRYPT option applies only to this option range since the length of the suboption cannot be determined by the receiver. In other cases, where the length is known, the receiver skips to the next suboption.

Suboptions with opcodes from 0xc0 to 0xff also contain an opcode-specific length of data. As before, these suboptions can be either fixed length or variable length. However, suboptions in this range are classed as mandatory as far as the protocol is concerned. However, they are not MANDATORY to implement unless otherwise stated, as discussed below.

```

bit  0 1 2 3 4 5 6 7
    +---+---+---+---+---+---+---+---+...
    |           Opcode =         | data
    | 1 1 x x x x x x |
    +---+---+---+---+---+---+---+---+...

```

Should a host encounter an unknown opcode greater than or equal to 0xc0 during the setup phase of the protocol, the host MUST transition to the DISABLED state. It SHOULD respond with both a DECLINE suboption and an UNKNOWN suboption specifying the opcode of the unknown mandatory suboption, after which the host MUST NOT send any further CRYPT options.

Should a host encounter an unknown opcode greater than or equal to 0xc0 while in the ENCRYPTING state, the host MUST respond with an UNKNOWN suboption specifying the opcode of the unknown mandatory

suboption, and should ensure the session continues with the same encryption and authentication state as it had before the segment was received. This may require ignoring other suboptions within the same message, or reverting any half-negotiated state.

Table 4 summarizes the opcodes discussed in this document. It is MANDATORY that all implementations support every opcode in this table. Each opcode is listed with the length in bytes of the suboption (including the opcode byte), or * for variable-length suboptions. The last column specifies in which of the (S)etup phase, (E)NCRYPTING state, and (D)ISABLED state an opcode may be used. A host MUST NOT send an option unless it is in one of the stages indicated by this column.

Value	Length	Name	Stages
0x01	1	HELLO	S
0x02	1	HELLO-app-support	S
0x03	1	HELLO-app-mandatory	S
0x04	1	DECLINE	SD
0x05	1	NEXTK2	S
0x06	1	NEXTK2-app-support	S
0x07	1	INIT1	S
0x08	1	INIT2	S
0x41	*	PKCONF	S
0x42	*	PKCONF-app-support	S
0x43	*	UNKNOWN	SED
0x44	*	SYNCOOKIE	S
0x45	*	ACKCOOKIE	SED
0x80	5	SYNC_REQ	E
0x81	5	SYNC_OK	E
0x82	2	REKEY	E
0x83	6	REKEYSTREAM	E
0x84	10	NEXTK1	S
0x85	*	IV	E

Opcodes for suboptions of the TCP CRYPT option.

Table 4

If a TCP segment (sent by an active opener) has the SYN flag set, the ACK flag clear, and one or more TCP CRYPT options, there is an implicit HELLO suboption even if that suboption does not appear in the segment. In particular, when such a SYN segment contains a single, empty, two-byte TCP CRYPT option, the passive opener MUST interpret that option as equivalent to the three-byte TCP option

composed of bytes OPT1, 3, 1 (Kind = OPT1, Length = 3, Suboption = HELLO).

A host MUST enter the DISABLED state if, during the setup phase, it receives a segment containing neither a TCP CRYPT nor a TCP MAC option. This is for robustness against middleboxes that strip options. A host MUST also enter DISABLED if, during the setup phase, it receives a DECLINE suboption or any unrecognized suboption with opcode greater than or equal to 0xc0. The DECLINE option is the preferred way for a host to refuse tcpcrypt. A host MAY also choose reply without a TCP CRYPT option to disable tcpcrypt. Once a host has entered DISABLED, it MUST NOT include the MAC option in any transmitted segment. The host MAY include a CRYPT option in the next segment transmitted, but only if the segment also contains the DECLINE suboption. All subsequently transmitted packets MUST NOT contain the CRYPT option.

4.3.1. The HELLO suboption

The HELLO dataless suboption MUST only appear in a segment with the SYN control bit set. It is used by an active opener to indicate interest in using tcpcrypt for a connection, and by a passive opener to indicate that the passive opener wishes to play the "C" role.

The initial SYN segment from an active opener wishing to use tcpcrypt MUST contain a TCP CRYPT option with either an explicit or an implicit HELLO suboption.

After receiving a SYN segment with the HELLO suboption, a passive opener MUST respond in one of three ways:

- o To continue setting up tcpcrypt and play the "S" role, the passive opener MUST respond with a PKCONF suboption in the SYN-ACK segment and transition to S-MODE.
- o To continue setting up tcpcrypt and play the "C" role, the passive opener MUST respond with a HELLO suboption in the SYN-ACK segment and transition to HELLO-SENT.
- o To continue without tcpcrypt, the passive opener MUST respond with either no CRYPT option or the DECLINE suboption in the SYN-ACK segment, then transition to the DISABLED state.

An active opener receiving HELLO in a SYN-ACK segment must either transition to S-MODE and respond with a PKCONF suboption, or transition to DISABLED.

There are three variants of the HELLO option used for application-

level authentication, each encoded differently as shown in Table 4. The variants are: a plain HELLO where the application is not tcpcrypt-aware (but the kernel is), an "application supported" HELLO where the application is tcpcrypt-aware and is advertising the fact, and a "application mandatory" HELLO where the application requires the remote application to support tcpcrypt otherwise the connection MUST revert to plain TCP. The application supported HELLO can be used, for example, when implementing HTTP digest authentication - an application can check whether the peer's application is tcpcrypt aware and proceed to authenticate tcpcrypt's session ID over HTTP, otherwise reverting to standard HTTP digest authentication. The application mandatory HELLO can be used, for example, when implementing an SSL library that attempts tcpcrypt but reverts to SSL if the peer's SSL library does not support tcpcrypt. The application mandatory HELLO avoids double encrypting (SSL-over-tcpcrypt) since the connection will revert to plain TCP if the remote SSL library is not tcpcrypt-ware.

4.3.2. The DECLINE suboption

The DECLINE dataless suboption is sent by a host to indicate that the host will not enable tcpcrypt on a connection. If a host is in the DISABLED state or transitioning to the DISABLED state, and the host transmits a segment containing a CRYPT option, then the segment MUST contain the DECLINE suboption.

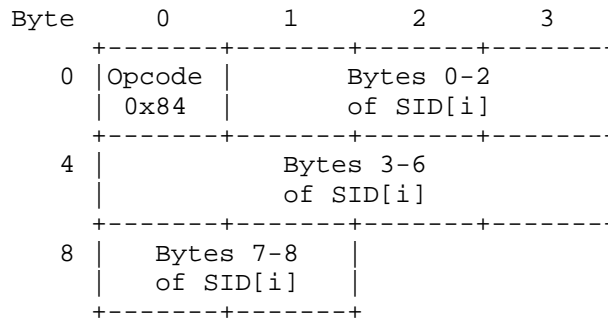
A passive opener SHOULD send a DECLINE suboption in response to a HELLO suboption or NEXTK1 suboption in a received SYN segment if it supports tcpcrypt but does not wish to engage in encryption for this particular session.

Implementations MUST NOT send segments containing the DECLINE suboption from the C-MODE or ENCRYPTING states.

4.3.3. The NEXTK1 and NEXTK2 suboptions

The NEXTK1 suboption MUST only appear in a segment with the SYN control bit set and the ACK bit clear. It is used by the active opener to initiate a TCP session without the overhead of public key cryptography. The new session key is derived from a previously negotiated session secret, as described in Section 3.8.

The suboption is always 10 bytes in length; the data contains the first nine bytes of SID[i] and is used to to start the session with session secret ss[i]. The format of the suboption is:



Format of the NEXTK1 suboption

The active opener MUST use the lowest value of *i* that has not already appeared in a NEXTK1 segment exchanged with the same host and for the same pre-session seed.

If the passive opener recognizes SID[i] and knows ss[i], it SHOULD respond with a segment containing the dataless NEXTK2 suboption. The NEXTK2 option MUST only appear in a segment with both the SYN and ACK bits set.

If the passive opener does not recognize SID[i], or SID[i] is not valid or has already been used, the passive opener SHOULD respond with a PKCONF or HELLO option and continue key negotiation as usual.

When two hosts have previously negotiated a tcpcrypt session, either host may use the NEXTK1 option regardless of which host was the active opener or played the "C" role in the previous session. However, a given host must either encrypt with *k_{cs}* for all sessions derived from the same pre-session seed, or *k_{sc}*. Thus, which keys a host uses to send segments depends only whether the host played the "C" or "S" role in the initial session that used ss[0]; it is not affected by which host was the active opener transmitting the SYN segment containing a NEXTK1 suboption.

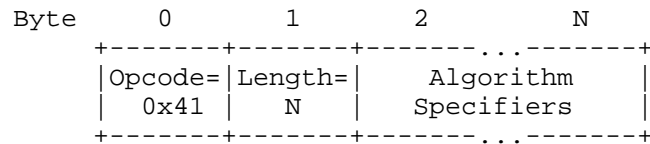
A host MUST reject a NEXTK1 message if it has previously sent or received one with the same SID[i]. In the event that two hosts simultaneously send SYN segments to each other with the same SID[i], but the two segments are not part of a simultaneous open, both connections will have to revert to public key cryptography. To avoid this limitation, implementations MAY chose to implement session caching such that a given pre-session key is only good for either passive or active opens at the same host, not both.

In the case of simultaneous open, two hosts that simultaneously send SYN packets with NEXTK1 and the same SID[i] may establish a

connection, as described in Section 4.2.1.

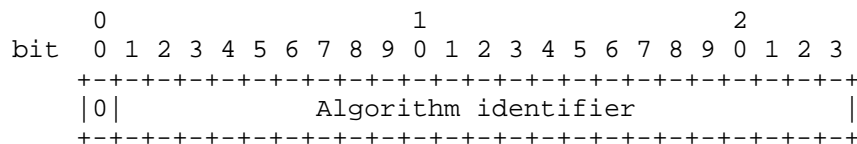
4.3.4. The PKCONF suboption

The PKCONF option has the following format:



Format of the PKCONF suboption

The suboption data, whose length (N-2) must be divisible by 3, contains one or more 3-byte algorithm specifiers of the following form:



Format of algorithm specifier within PKCONF. Fields starting with 1 are reserved for future use by algorithm identifiers longer than three bytes.

The algorithm identifier specifies a number of parameters, defined in Figure 3.

Hosts MUST implement OAEP+-RSA3 and ECDHE-P256 and ECDHE-P512.

Servers demanding utmost performance SHOULD use RSA because the RSA encrypt operation is much faster than Diffie-Hellman operations, resulting in a higher connection rate.

Depending on the encoding of the PKCONF suboption (see Table 4), it can indicate whether "S's" application is tcpcrypt-aware or not. For the "C" role, the encoding of the HELLO suboption does this. This mechanism can be used for bootstrapping application-level authentication without requiring probing in upper layer protocols to check for support (which may not be possible). The application controls these encodings via the TCP_CRYPT_SUPPORT socket option.

4.3.5. The UNKNOWN suboption

The UNKNOWN option has the following format:

Byte	0	1	2	N
	+	+	+	+
	Opcode=	Length=	N-2 unknown one-byte	
	0x42	N	opcodes received	
	+	+	+	+

Format of the UNKNOWN suboption

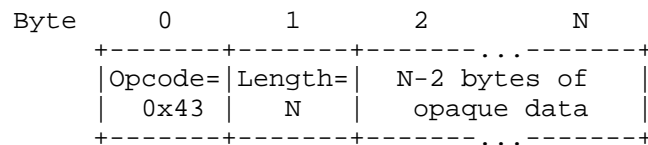
This suboption is sent in response to an unknown suboption that has been received. The contents of the option are a complete list of the mandatory suboption opcodes from the received packet that were not understood. Note that this option is only sent once, in the next packet that the host sends. This means that it is reliable when sent in a SYN-ACK, but unreliable otherwise. Any mechanism sending new mandatory attributes must take this into account. If multiple packets, each containing unknown options, are received before an UNKNOWN suboption can be sent, the options list **MUST** contain the union of the two sets. The order of the opcode list is not significant.

If a host receives an unknown option, it **SHOULD** reply with the UNKNOWN suboption to notify the other side. If the host transitions to **DISABLED** as a result of the unknown option, then the host **MUST** also include the **DECLINE** suboption if it sends an UNKNOWN suboption (or more generally if it includes a **CRYPT** option in the next packet).

As a special case, if **PKCONF** (0x41) or **INIT1** (0x06) appears in the unknown opcode list, it does not mean the sender does not understand the option (since these options are **MANDATORY**). Instead, it means the sender does not implement any of the algorithms specified in the **PKCONF** or **INIT1** message. In either case, the segment must also contain a **DECLINE** suboption.

4.3.6. The SYNCOOKIE and ACKCOOKIE suboptions

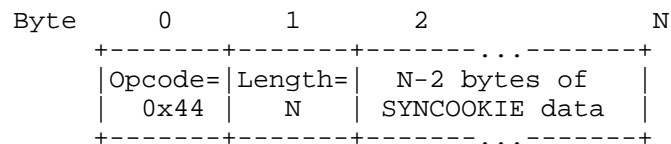
A passive opener **MAY** include the **SYNCOOKIE** suboption in a segment with both the **SYN** and **ACK** flags set. **SYNCOOKIE** allows a server to be stateless until the TCP handshake has completed. It has the following format:



Format of the SYNCOOKIE suboption

The data is opaque as far as the protocol is concerned; it is entirely up to implementations how to make use of this suboption to hold state. It is OPTIONAL to send a SYNCOOKIE, but MANDATORY to understand and respond to them.

The ACKCOOKIE suboption echoes the contents of a SYNCOOKIE; it MUST be sent in a packet acknowledging receipt of a packet containing a SYNCOOKIE, and MUST NOT be sent in any other packet. It has the following format:



Format of the ACKCOOKIE suboption

Servers that rely on suboption data from ACKCOOKIE to reconstruct session state SHOULD embed a cryptographically strong message authentication code within the SYNCOOKIE data so as to be able to reject forged ACKCOOKIE suboptions.

Though an implementation MUST NOT send a SYNCOOKIE in any context except the SYN-ACK packet returned by a passive opener, implementations SHOULD accept SYNCOOKIES in other contexts and reply with the appropriate ACKCOOKIE if possible.

4.3.7. The SYNC_REQ and SYNC_OK suboptions

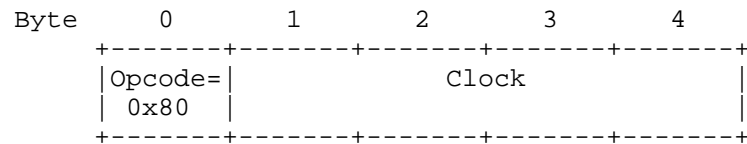
Many hosts implement TCP Keep-Alives [RFC1122] as an option for applications to ensure that the other end of a TCP connection still exists even when there is no data to be sent. A TCP Keep-Alive segment carries a sequence number one prior to the beginning of the send window, and may carry one byte of "garbage" data. Such a segment causes the remote side to send an acknowledgment.

Unfortunately, Keep-Alive acknowledgments might not contain unique data. Hence, an old but cryptographically valid acknowledgment could be replayed by an attacker to prolong the existence of a session at

one host after the other end of the connection no longer exists. (Such an attack might prevent a process with sensitive data from exiting, giving an attacker more time to compromise a host and extract the sensitive data.)

The TCP Timestamps Option (TSopt) [RFC1323] could alternatively have been used to make Keep-Alives unique. However, because some middleboxes change the value of TSopt in packets, tcpcrypt does not protect the contents of the TCP TSopt option. Hence the SYNC_REQ and SYNC_OK suboptions allow the cryptographically protected TCP CRYPT option to contain unique data.

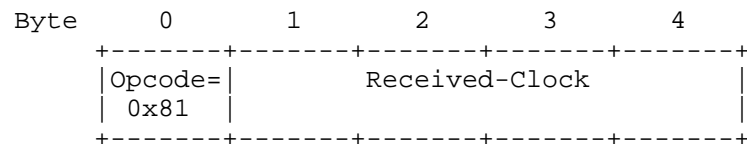
The SYNC_REQ suboption is always 5 bytes, and has the following format:



Format of the SYNC_REQ suboption

Clock is a 32-bit non-decreasing value. A host **MUST** increment Clock at least once for every interval in which it sends a Keep-Alive. Implementations that support TSopt **MAY** chose to use the same value for Clock that they would put in the TSval field of the TCP TSopt. However, implementations **SHOULD** "fuzz" any system clocks used to avoid disclosing either when a host was last rebooted or at what rate the hardware clock drifts.

A host that receives a SYNC_REQ suboption **MUST** reply with a SYNC_OK suboption, which is always five bytes and has the following format:



Format of the SYNC_OK suboption

The value of Received-Clock depends on the values of the Clock fields in SYNC_REQ messages a host has received. A host must set Received-Clock to a value at least as high as the most recently received Clock, but no higher than the highest Clock value received this session. If a host delays acknowledgment of multiple packets with SYNC_REQ suboptions, it **SHOULD** send a single SYNC_OK with Received-

Clock set to the highest Clock in the packets it is acknowledging.

Because middleboxes sometimes "correct" inconsistent retransmissions, Keep-Alive segments with one byte of garbage data MUST use the same ciphertext byte as previously transmitted for that sequence number. Otherwise, a middlebox might change the byte back to its value in the original transmission, causing the cryptographic MAC to fail.

4.3.8. The REKEY and REKEYSTREAM suboptions

The REKEY and REKEYSTREAM suboptions are used to evolve encryption keys. Exactly one of the two options is valid with any given symmetric encryption algorithm and mode. Generally block ciphers will use REKEY while stream ciphers use REKEYSTREAM. We refer to a segment containing either option as a REKEY segment.

REKEY allows hosts to wipe from memory keys that could decrypt previously transmitted segments. It also allows the use of message authentication codes that are only secure up to a fixed number of messages. However, implementations MUST work in the presence of middleboxes that "correct" inconsistent data retransmissions. Hence, the value of ciphertext bytes must be the same in the original transmission and all retransmissions of a particular sequence number. This means a host MUST always use the same encryption key when transmitting or retransmitting the same range of sequence numbers. Re-keying only affects data transmitted in the future. Moreover, segments encrypted with different keysets MUST NOT be combined in retransmissions.

When switching keys, the REKEY suboption specifies which key set has been used to encrypt and integrity-protect the current segment. The suboption is always two bytes, and has the following format:

Byte	0	1
	+-----+-----+	
	Opcode=	KeyLSB
	0x83	
	+-----+-----+	

Format of the REKEY suboption

KeyLSB is the generation number of the keys used to encrypt and MAC the current segment, modulo 256. REKEYSTREAM is the same as REKEY but includes the TCP Sequence Number offset at which the key change took effect, for cases in which decryption requires knowing how many bytes have been encrypted thus far with a key. To interoperate with middleboxes that rewrite sequence numbers, offsets from the Initial Sequence Number (ISN) are used instead of TCP sequence numbers

throughout tcpcrypt. The same occurs when dealing with acknowledgement numbers.

Byte	0	1	2	3	4	5
	+	+	+	+	+	+
	Opcode=	KeyLSB		Sequence Number	Offset	
	0x83			from ISN		
	+	+	+	+	+	+

Format of the REKEYSTREAM suboption

A host MAY use REKEY to increment the session key generation number beyond the highest generation it knows the other side to be using. We call this process `_initiating_` re-keying. When one host initiates re-keying, the other host MUST increment its key generation number to match, as described below (unless the other host has also simultaneously initiated re-keying).

A host MAY initiate re-keying by including a REKEY suboption in a `_syncable_` segment. A syncable segment is one that either contains data, or is acknowledgment-only but contains a SYNC_REQ suboption with a fresh Clock value--i.e., higher than any Clock value it has previously transmitted. We say a syncable segment is `_synced_` when the transmitter knows the remote side has received it and all previous sequence numbers. A data segment is synced when the transmitter receives a cumulative acknowledgment for its sequence number (a Selective Acknowledgment [RFC2018] is insufficient). An acknowledgment-only segment is synced when the sender receives an acknowledgment for its sequence number and a SYNC_OK with a high enough Clock number.

A host MUST NOT initiate re-keying with an acknowledgment-only segment that has either no SYNC_REQ suboption or a SYNC_REQ with an old Clock value, because such a segment is not syncable. A host MUST NOT initiate re-keying with any KeyLSB other than its current key number plus one modulo 256.

When a host receives a segment containing a REKEY suboption, it MUST proceed as follows:

1. The receiver computes RECEIVE-KEY-NUMBER to be the closest integer to its own transmit key number that also equals KeyLSB modulo 256. If no number is closest (because KeyLSB is exactly 128 away from the transmit number modulo 256), the receiver MUST discard the segment. If RECEIVE-KEY-NUMBER is negative, the receiver MUST also discard the segment.

2. The receiver MUST authenticate and decrypt the segment using the receive keys with generation number RECEIVE-KEY-NUMBER. The receiver MUST discard the packet as usual if the MAC is invalid.
3. If RECEIVE-KEY-NUMBER is greater than the receiver's current transmit key number, the receiver must wait to receive all sequence numbers prior to the REKEY segment's. Once it receives segments covering all these missing sequence numbers (if any), it MUST increase its transmit number to RECEIVE-KEY-NUMBER and transmit a REKEY suboption. If the receiver has gotten multiple REKEY segments with different KeyLSB values, it MUST increase its transmit key number to the highest RECEIVE-KEY-NUMBER of any segment for which it is not missing prior sequence numbers.

After sending a REKEY (whether initiating re-keying or just responding), a host MUST continue to send REKEY in all subsequent segments until at least one of the following holds:

- o One of the REKEY segments the host transmitted for its current transmit key number was syncable, and it has been synced.
- o The host receives a cumulative acknowledgment for one of its REKEY segments with the current transmit key number, and the cumulative acknowledgment is in a segment encrypted with the new key but not containing a REKEY suboption.

A host SHOULD erase old keys from memory once the above requirements are met.

A host MUST NOT initiate re-keying if it initiated a re-keying less than 60 seconds ago and has not transmitted at least 1 Megabyte (increased its sequence number by 1,048,576) since the last re-keying. A host MUST NOT initiate re-keying if it has outstanding unacknowledged REKEY segments for key numbers that are 127 or more below the current key. A host SHOULD not initiate more than one concurrent re-key operation if it has no data to send.

4.3.9. The INIT1 and INIT2 suboptions

The INIT1 dataless suboption indicates that the Data portion of the TCP segment contains the following data structure:

Byte	0	1	2	3
	+-----+-----+-----+-----+			
	# bytes INIT1			
	+-----+-----+-----+-----+			
	INIT1_MAGIC # auth enc alg			
	+-----+-----+-----+-----+			
	# bytes of N_C # byte of PK_C			
	+-----+-----+-----+-----+			
	authenticated encryption			
	: algorithms :			
	+-----+-----+-----+-----+			
	N_C			
	: :			
	+-----+-----+-----+-----+			
	0 type of K_C			
	+-----+-----+-----+-----+			
	PK_C			
	: :			
	+-----+-----+-----+-----+			

The INIT1_MAGIC is specified in Table 7. The following values for authenticated sequence mode (ASM) encryption algorithms are defined:

The first entry is mandatory and MUST be supported by all implementations. The sequence number for ASM mode is TCP's extended 64-bit sequence number offset from the ISN.

The value "type of PK_C" must be one of the public key specifiers included earlier in the other host's PKCONF message.

The INIT2 dataless suboption indicates that the Data portion of the TCP segment contains the following data structure:

Byte	0	1	2	3
	+-----+-----+-----+-----+			
	# byte INIT2			
	+-----+-----+-----+-----+			
	INIT2_MAGIC #byte kmaterial			
	+-----+-----+-----+-----+			
	symmetric cipher suite			
	+-----+-----+-----+-----+			
	key material			
	: :			
	+-----+-----+-----+-----+			

Format of the INIT2 suboption

Figure 2

The INIT2_MAGIC is specified in Table 7. The symmetric cipher suite is one selected by the host transmitting the INIT2 segment, which will be playing the "S" role. The key material depends on the public key cipher selected, as described in Section 3.4. When ECDHE is used, key material is encoded as follows:

Byte	0	1	N	M
	+-----+-----+-----+-----+			
	# byte	N_S	PK_S	
	N_S			
	+-----+-----+-----+-----+			

Hosts MUST set the TCP PSH control bits on INIT1 and INIT2 segments. Implementations MUST NOT set the TCP FIN control bit on INIT segments.

4.3.10. The IV suboption

The IV suboption is used to hold an initialization vector (IV) when the negotiated encryption mode requires an initialization vector to be transmitted with packets. It MUST NOT be included in transmitted packets except in the ENCRYPTING state when the negotiated encryption mode requires IVs. When the negotiated encryption mode does require IVs, all segments transmitted in ENCRYPTING mode MUST contain an IV suboption.

The IV suboption has the following format:

Byte	0	1	N
	+-----+-----+-----+		
	Opcode=	Initialization	
	0x85	Vector	
	+-----+-----+-----+		

Format of the IV suboption

The length N of the IV is determined by the encryption algorithm and mode negotiated.

As discussed in Section 4.3.8, a host MUST always transmit the same ciphertext byte in retransmissions of a particular sequence number. Thus, retransmitted segments must use the same IV each time. Moreover, previously transmitted segments MUST NOT be combined on retransmission if their IVs would prevent the ciphertext bytes from remaining the same as in the original transmission.

4.4. The TCP MAC option

The MAC option is used to authenticate a TCP segment. Once a host has entered the encrypting phase for a session, the HOST must include a TCP MAC option in all segments it sends. Furthermore, once in the encrypting phase, a host MUST ignore any segments it receives that do not have a valid MAC option, except for segments with the RST bit set if the application has not requested cryptographic verification of RST segments.

The length of the MAC option is determined by the symmetric message authentication code selected. The format of the MAC option is:

Byte	0	1	2	N+1
	+	+	+	+
	Kind	Len=	N-byte	
	OPT2	2+N	MAC	
	+	+	+	+

Format of TCP MAC option

The MAC is the authentication tag as output from authenticated encryption. Apart from payload, two headers are included in the authenticated encryption process: a pseudo-header structure we call Assoc-Data, and an acknowledgment structure we call Up-Data. The format of Assoc-Data is as follows:

Byte	0	1	2	3
	+	+	+	+
0	0x8000		length	
	+	+	+	+
4	off	flags	window	
	+	+	+	+
8	0x0000		urg	
	+	+	+	+
12		seqno offset hi		
	+	+	+	+
16		seqno offset		
	+	+	+	+
20		options		
:	:	:	:	:
	+	+	+	+

Assoc-Data data structure

The fields of Assoc-Data are defined as follows:

length

Total size of the TCP segment from the start of the TCP header to the end of the IP datagram.

off

Byte 12 of the TCP header (Data Offset)

flags

Byte 13 of the TCP header (Control Bits)

window

Bytes 14-15 of the TCP header (Window)

urg

Bytes 18-19 of the TCP header (Urgent Pointer)

seqno offset hi

Number of times the seqno offset field has wrapped from 0xffffffff -> 0

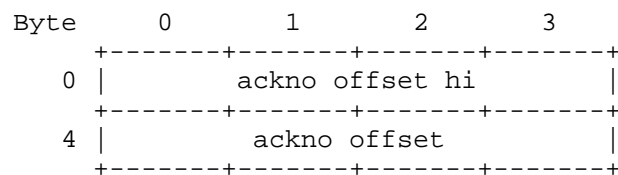
seqno offset

The low 32 bits of the sequence number offset (the Sequence Number in the TCP header - ISN)

options

These are bytes 20-off of the TCP header. However, where the TSOPT (8), Skeeter (16), Bubba (17), MD5 (19), and MAC (OPT2) options appear, their contents (all but the kind and length bytes) are replaced with all zeroes.

The format of the Up-Data structure is as follows:



Up-Data data structure

The fields of Up-Data are defined as follows:

ackno offset hi The number of times ackno offset hi has wrapped from 0xffffffff -> 0.

ackno offset The lower 32 bits of the acknowledgement number offset from the remote end's ISN (TCP's acknowledgement header - ISN received).

The two structures, Assoc-Data and Up-Data, are used in ASM mode to calculate the TCP MAC option.

5. Examples

To illustrate these suboptions, consider the following series of ways in which a TCP connection may be established from host A to host B. We use notation S for SYN-only packet, SA for SYN-ACK packet, and A for packets with the ACK bit but not SYN bit. These examples are not normative.

5.1. Example 1: Normal handshake

```
(1) A -> B: S    CRYPT<>
(2) B -> A: SA  CRYPT<PKCONF<0x200,0x201>>
(3) A -> B: A    data<INIT1...>
(4) B -> A: A    data<INIT2...>
(5) A -> B: A    MAC<m> data<...>
```

(1) A indicates interest in using tcpcrypt. In (2), the server indicates willingness to use ECDHE with curves P256 and P512. Messages (3) and (4) complete the INIT1 and INIT2 key exchange messages described above, which are embedded in the data portion of the TCP segment. (5) From this point on, all messages are encrypted and their integrity protected by a MAC option.

5.2. Example 2: Normal handshake with SYN cookie

```
(1) A -> B: S    CRYPT<>
(2) B -> A: SA  CRYPT<PKCONF<0x200,0x201>, SYNCOOKIE<val>>
(3) A -> B: A    CRYPT<ACKCOOKIE<val>> data<INIT1...>
(4) B -> A: A    data<INIT2...>
(5) B -> A: A    MAC<m> data<...>
```

Same as previous example, except the server sends the client a SYN cookie value, which the client must echo in (3). Here also the application level protocol begins by B transmitting data, while in the previous example, A was the first to transmit application-level data.

5.3. Example 3: tcpcrypt unsupported

```
(1) A -> B: S  CRYPT<>
(2) B -> A: SA
(3) A -> A: A
```

(1) A indicates interest in using tcpcrypt. (2) B does not support tcpcrypt, or a middle box strips out the CRYPT TCP option. (3) the client completes a normal three-way handshake, and tcpcrypt is not enabled for the connection.

5.4. Example 4: Reusing established state

```
(1) A -> B: S  CRYPT<NEXTK1<ID>>
(2) B -> A: SA CRYPT<NEXTK2>
(3) A -> A: A  MAC<m>
```

(1) A indicates interest in using tcpcrypt with a session key derived from an existing key, to avoid the use of public key cryptography for the new session. (2) B supports tcpcrypt, has ID in its session ID cache, and is willing to proceed with session caching. (3) the client completes tcpcrypt's handshake within TCP's three-way handshake and tcpcrypt is enabled for the connection.

5.5. Example 5: Decline of state reuse

```
(1) A -> B: S  CRYPT<NEXTK1<ID>>
(2) B -> A: SA CRYPT<PKCONF<1, 4, 16>>
(3) A -> B: A  data<INIT1...>
(4) B -> A: A  data<INIT2...>
(5) A -> B: A  MAC<m> data<...>
```

A wishes to use a key derived from a previous session key, but B does not recognize the session ID or has flushed it from its cache. Therefore, session establishment proceeds as in the first connection, using public key cryptography to negotiate a new series of session secrets (ss[i] values).

5.6. Example 6: Reversal of client and server roles

```
(1) A -> B: S  CRYPT<>
(2) B -> A: SA CRYPT<HELLO>
(3) A -> B: A  CRYPT<PKCONF<0x100>>
(4) B -> A: A  data<INIT1...>
(5) A -> B: A  data<INIT2...>
(6) B -> A: A  MAC<m> data<...>
```

Here the passive opener, B, wishes to play the role of the decryptor

using RSA. By sending a HELLO suboption, B causes A to switch roles, so that now A is "S" and B plays the role of "C".

6. API extensions

The getsockopt call should have new options for IPPROTO_TCP:

TCP_CRYPT_SESSID -> returns the session ID and MUST return an error if tcpcrypt is in not in the ENCRYPTING state (e.g., because it has transitioned to DISABLED).

TCP_CRYPT_CMODE -> returns 1 if the local host played the "C" role in session key negotiation, 0 otherwise.

TCP_CRYPT_PUBKEY_LOCAL -> When the local host played the "C" role, returns the hosts public key, PK_C. When the local host played the "S" role, returns PK_S if KX_S supports such a value or returns an error otherwise. Hosts MAY return an error after transmitting the first application-level payload bytes (soas to reclaim the memory used to store keys).

TCP_CRYPT_PUBKEY_PEER -> Analogous to TCP_CRYPT_PUBKEY_LOCAL with the roles reversed. (Returns PK_C when the local host played the "S" role, and PK_S, if applicable, when the local host played the "C" role.)

TCP_CRYPT_CONF -> returns the four-byte authenticated encryption algorithm in use by the connection (as specified in Table 6). In addition, implementations SHOULD provide the three-byte public key cipher (Figure 3) initially used to negotiate the session keys, as well as the public key length for algorithms with variable key sizes (e.g., OAEP+-RSA3).

TCP_CRYPT_SUPPORT -> returns 1 if the remote application is tcpcrypt-aware, as indicated by the remote host's use of a HELLO-app-support, HELLO-app-mandatory, or PKCONF-app-support CRYPT suboption (see Table 4).

The setsockopt call should have:

TCP_CRYPT_CACHE_FLUSH -> setting this option to non-zero wipes cached session keys. Useful if application-level authentication discovers a man in the middle attack, to prevent the next connection from using NEXTK.

The following options should be readable and writable with getsockopt and setsockopt:

TCP_CRYPT_ENABLE -> one bit, enables or disables tcpcrypt extension on an unconnected (listening or new) socket.

TCP_CRYPT_SECURST -> one bit, means ignore unauthenticated RST packets for this connection when set to 1.

TCP_CRYPT_CMODE_{DEFAULT,NEVER,ALWAYS}[_NK] -> As described in Section 4.2.

TCP_CRYPT_PKCONF -> set of allowed public key algorithms and CPRFs this host advertises in CRYPT PKCONF suboptions.

TCP_CRYPT_CCONF -> set of allowed symmetric ciphers and message authentication codes this host advertises in CRYPT INIT1 segments.

TCP_CRYPT_SCONF -> order of preference of symmetric ciphers.

TCP_CRYPT_SUPPORT -> set to 1 if the application is tcpcrypt-aware. set to 2 if the application is tcpcrypt-aware and wishes to enter the DISABLED state if the remote application is not tcpcrypt-aware. An active opener SHOULD set the default value to 0 for each new connection. A passive opener SHOULD use a default value to 0 for each port, but SHOULD inherit the value of the listening socket for accepted connections. The behavior for each value is as follows:

When set to 0 The host MUST transition to the DISABLED state upon receiving a HELLO-app-mandatory option. The host MUST NOT send the HELLO-app-support, HELLO-app-mandatory, NEXTK2-app-support, or PKCONF-app-support options.

When set to 1 The "C" role host MUST use HELLO-app-support in place of the HELLO option, while the "S" role host MUST use the "PKCONF-app-support" in place of the "PKCONF" option. Either role must use NEXTK2-app-support in place of NEXTK2.

When set to 2 The "C" role host MUST use HELLO-app-mandatory option in place of the HELLO option, while the "S" role host MUST use "PKCONF-app-support" in place of the "PKCONF" option. Either role must use NEXTK2-app-support in place of NEXTK2. Either host MUST transition to DISABLED upon receipt of a HELLO or PKCONF option, but MUST proceed as usual in response to HELLO-app-support, HELLO-app-mandatory, and PKCONF-app-support.

Finally, system administrators must be able to set the following system-wide parameters:

- o Default TCP_CRYPT_ENABLE value
- o Default TCP_CRYPT_PKCONF value
- o Default TCP_CRYPT_CCONF value
- o Default TCP_CRYPT_SCONF value
- o Types, key lengths, and regeneration intervals of local host's short-lived public keys

The session ID can be used for end-to-end security. For instance, applications might sign the session ID with public keys to authenticate their ends of a connection. Because session IDs are not secret, servers can sign them in batches to amortize the cost of the signature over multiple connections. Alternatively, DSA signatures are cheaper to compute than to verify, so might be a good way for servers to authenticate themselves. A voice application could display the session ID on both parties' screens, and if they confirm by voice that they have the same ID, then the conversation is secure.

Because the public key may change less often than once a session, it may alternatively be useful for the local end of a connection to authenticate itself by signing the local host's public key instead of the session ID.

7. Acknowledgments

This work was funded by gifts from Intel (to Brad Karp) and from Google, and by NSF award CNS-0716806 (A Clean-Slate Infrastructure for Information Flow Control).

8. IANA Considerations

The following numbers need assignment by IANA:

- o New TCP option kind number for CRYPT
- o New TCP option kind number for MAC

A new registry entitled "tcpcrypt CRYPT suboptions" needs to be maintained by IANA as per the following table.

Symbol	Value
HELLO	0x01
HELLO-app-support	0x02
HELLO-app-mandatory	0x03
DECLINE	0x04
NEXTK2	0x05
NEXTK2-app-support	0x06
INIT1	0x07
INIT2	0x08
PKCONF	0x41
PKCONF-app-support	0x42
UNKNOWN	0x43
SYNCOOKIE	0x44
ACKCOOKIE	0x45
SYNC_REQ	0x80
SYNC_OK	0x81
REKEY	0x82
REKEYSTREAM	0x83
NEXTK1	0x84
IV	0x85

TCP CRYPT suboptions.

Table 5

A "tcpcrypt Algorithm Identifiers" registry needs to be maintained by IANA as per the following table.

Algorithm Identifier	Value
Cipher: OAEP+-RSA with exponent 3 Extract: HKDF-Extract-SHA256 CPRF: HKDF-Expand-SHA256 N_C len: 32 bytes R_S len: 48 bytes K_LEN: 32 bytes	0x000100
Cipher: ECDHE-P256 Extract: HKDF-Extract-SHA256 CPRF: HKDF-Expand-SHA256 N_C len: 32 bytes N_S len: 32 bytes K_LEN: 32 bytes	0x000200
Cipher: ECDHE-P512 Extract: HKDF-Extract-SHA256 CPRF: HKDF-Expand-SHA256 N_C len: 32 bytes N_S len: 32 bytes K_LEN: 32 bytes	0x000201

TCP CRYPT algorithm identifiers.

Figure 3

A "tcpcrypt authenticated encryption algorithms" registry needs to be maintained by IANA as per the following table.

Authenticated Encryption	value
AES-128 ASM mode HMAC-SHA2-128 AES-128 ACK MAC	0x00000100
AES-128 ASM mode Poly1305-AES AES-128 ACK MAC	0x00000200
AES-128 ASM mode CMAC-AES-128 AES-128 ACK MAC	0x00000300

TCP CRYPT authenticated encryption algorithms.

Table 6

9. Security Considerations

Tcpcrypt guarantees that no man-in-the-middle attacks occurred if Session IDs match on both ends of a connection, unless the attacker has broken the underlying cryptographic primitives (e.g., RSA). A proof has been published [tcpcrypt].

If the application performs no authentication, then there are no guarantees against active attackers. Session IDs can be logged on both ends and man-in-the-middle attacks can be detected after the fact by comparing Session IDs offline.

Session IDs are not confidential.

Tcpcrypt can be downgraded to regular TCP during the connection setup phase by removing any of the CRYPT options. The downgrade, and absence of protection, can of course be detected by the application as no Session ID will be returned.

By default tcpcrypt does not protect against RST packet injection. The connection must be configured with TCP_CRYPT_RSTCHK enabled to protect against malicious (unMACed) RSTs.

tcpcrypt uses short-lived keys to provide some forward secrecy. If a key is compromised all connections (new and cached) derived from that key will be compromised. The life of these keys should be kept to a minimum for stronger protection. A life of less than two minutes is recommended. Keys can be generated as frequently as practical, for example when servers have idle CPU time. For ECDHE-based key agreement, a new key can be chosen for each connection.

In the 4-way handshake, tcpcrypt does not have a key confirmation step. Hence, an active attacker can cause a connection to hang, though this is possible even without tcpcrypt by altering sequence and ack numbers.

Attackers cannot force passive openers to move forward in their session caching chain without guessing the content of the NEXTK1 option, which will be hard without key knowledge.

10. References

10.1. Normative References

- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

- [RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [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.
- [RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2437] Kaliski, B. and J. Staddon, "PKCS #1: RSA Cryptography Specifications Version 2.0", RFC 2437, October 1998.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, January 2013.

10.2. Informative References

- [I-D.narten-iana-considerations-rfc2434bis]
Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs",
draft-narten-iana-considerations-rfc2434bis-09 (work in progress), March 2008.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.
- [aggregate-macs]
Katz, J. and A. Lindell, "Aggregate Message Authentication Codes", Topics in Cryptology - CT-RSA , 2008.
- [tcpcrypt]
Bittau, A., Hamburg, M., Handley, M., Mazieres, D., and D. Boneh, "The case for ubiquitous transport-level encryption", USENIX Security , 2010.

Appendix A. Protocol constant values

Value	Name
0x01	CONST_NEXTK
0x02	CONST_SESSID
0x03	CONST_REKEY
0x04	CONST_KEY_C
0x05	CONST_KEY_S
0x06	CONST_KEY_ENC
0x07	CONST_KEY_MAC
0x08	CONST_KEY_ACK
0x2911	INIT1_MAGIC
0x8310	INIT2_MAGIC

Protocol constants.

Table 7

Authors' Addresses

Andrea Bittau
Stanford University
Department of Computer Science
353 Serra Mall, Room 288
Stanford, CA 94305
US

Phone: +1 650 723 8777
Email: bittau@cs.stanford.edu

Dan Boneh
Stanford University
Department of Computer Science
353 Serra Mall, Room 475
Stanford, CA 94305
US

Phone: +1 650 725 3897
Email: dabo@cs.stanford.edu

Mike Hamburg
Stanford University
Department of Computer Science
353 Serra Mall, Room 475
Stanford, CA 94305
US

Phone: +1 650 725 3897
Email: mike@shiftleft.org

Mark Handley
University College London
Department of Computer Science
University College London
Gower St.
London WC1E 6BT
UK

Phone: +44 20 7679 7296
Email: M.Handley@cs.ucl.ac.uk

David Mazieres
Stanford University
Department of Computer Science
353 Serra Mall, Room 290
Stanford, CA 94305
US

Phone: +1 415 490 9451
Email: dm@uun.org

Quinn Slack
Stanford University
Department of Computer Science
353 Serra Mall, Room 288
Stanford, CA 94305
US

Phone: +1 650 723 8777
Email: sqs@cs.stanford.edu

TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Intended status: Informational
Expires: August 16, 2014

M. Kuehlewind, Ed.
University of Stuttgart
R. Scheffenegger
NetApp, Inc.
B. Briscoe
BT
February 12, 2014

Problem Statement and Requirements for a More Accurate ECN Feedback
draft-ietf-tcpm-accecn-reqs-05

Abstract

Explicit Congestion Notification (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 feed this information back to the sender. ECN is specified for TCP in such a way that it can only feed back one congestion signal per Round-Trip Time (RTT). In contrast, ECN for other transport protocols, such as RTP/UDP and SCTP, is specified with more accurate ECN feedback. Recent new TCP mechanisms (like ConEx or DCTCP) need more accurate ECN feedback in the case where more than one marking is received in one RTT. This document specifies requirements for an update to the TCP protocol to provide more accurate ECN feedback.

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 16, 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	2
1.1. Terminology	3
2. Recap of Classic ECN and ECN Nonce in IP/TCP	4
3. Use Cases	5
4. Requirements	7
5. Design Approaches	10
5.1. Re-Definition of ECN/NS Header Bits	10
5.2. Using Other Header Bits	11
5.3. Using a TCP Option	12
6. Acknowledgements	12
7. IANA Considerations	12
8. Security Considerations	12
9. References	13
9.1. Normative References	13
9.2. Informative References	13
Appendix A. Ambiguity of the More Accurate ECN Feedback in DCTCP	14
Authors' Addresses	15

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 feed this information back 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). This is sufficient for pre-existing TCP congestion control mechanisms that perform only one reduction in sending rate per RTT, independent of the number of ECN congestion marks. But recently proposed or deployed mechanisms like Congestion Exposure (ConEx) [RFC6789] or Data Center TCP (DCTCP) [Ali10] need more accurate ECN feedback to work correctly in the case where more than one marking is received in any one RTT.

ECN is also defined for transport protocols beside TCP. ECN feedback as defined for RTP/UDP [RFC6679] provides a very detailed level of information, delivering individual counters for all four ECN

codepoints as well as lost and duplicate segments, but at the cost of high signaling overhead. ECN feedback for SCTP [I-D.stewart-tsvwg-sctpecn] delivers a counter for the number of CE marked segments between CWR chunks, but also comes at the cost of increased overhead.

Today, implementations of DCTCP already exist that alter TCP's ECN feedback protocol in proprietary ways (DCTCP was released in Microsoft Windows 8, and implementations exist for Linux and FreeBSD). The changes DCTCP makes to TCP are not currently the subject of any IETF standardization activity, and they omit capability negotiation, relying instead on uniform configuration across a across all hosts and network devices with ECN capability. A primary motivation for this document is to intervene before each proprietary implementation invents its own non-interoperable handshake, which could lead to de facto consumption of the few flags or codepoints that remain available for standardizing capability negotiation.

This document lists requirements for a robust and interoperable more accurate TCP/ECN feedback protocol that all implementations of new TCP extensions, like ConEx and/or DCTCP, can use. While a new feedback scheme should still deliver as much information as classic ECN, this document also clarifies what has to be taken into consideration in addition. Thus the listed requirements should be addressed in the specification of a more accurate ECN feedback scheme. A few solutions have already been proposed. Section 5 demonstrates how to use the requirements to compare them, by briefly sketching their high level design choices and discussing the benefits and drawbacks of each.

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].

We use the following terminology from [RFC3168] and [RFC3540]:

The ECN field in the IP header:

Not-ECT: the not ECN-Capable Transport codepoint,

CE: the Congestion Experienced codepoint,

ECT(0): the first ECN-Capable Transport codepoint, and

ECT(1): the second ECN-Capable Transport codepoint.

The ECN flags in the TCP header:

CWR: the Congestion Window Reduced flag,

ECE: the ECN-Echo flag, and

NS: ECN Nonce Sum.

In this document, the ECN feedback scheme as specified in [RFC3168] is called 'classic ECN' and any new proposal is called a 'more accurate ECN feedback' scheme. A 'congestion mark' is defined as an IP packet where the CE codepoint is set. A 'congestion episode' refers to one or more congestion marks that belong to the same overload situation in the network (usually during one RTT). A TCP segment with the acknowledgment flag set is simply called ACK.

2. Recap of Classic ECN and ECN Nonce in IP/TCP

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. A network node can set both bits simultaneously when it experiences congestion. This leads to the four codepoints (not-ECT, ECT(0), ECT(1), and CE) as listed above.

In the TCP header the first two bits in byte 14 are defined as ECN feedback for each half-connection. A TCP receiver signals the reception of a congestion mark using the ECN-Echo (ECE) flag in the TCP header. For reliability, the receiver continues to set the ECE flag on every ACK. To enable the TCP receiver to determine when to stop setting the ECN-Echo flag, the sender sets the CWR flag upon reception of an ECE feedback signal. This always leads to a full RTT of ACKs with ECE set. Thus the receiver cannot signal back any additional CE markings arriving within the same RTT.

The ECN Nonce [RFC3540] is an experimental addition to ECN that the TCP sender can use to protect itself against accidental or malicious concealment of CE-marked (or dropped) packets. This addition defines the last bit of byte 13 in the TCP header as the Nonce Sum (NS) flag. The receiver maintains a nonce sum that counts the occurrence of ECT(1) packets, and signals the least significant bit of this sum on the 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

However, as the ECN Nonce is a separate extension to ECN, even if a sender tries to protect itself with the ECN Nonce, any receiver wishing to conceal marked packets only has to pretend not to support the ECN Nonce and simply does not provide any nonce sum feedback.

An alternative for a sender to assure feedback integrity has been proposed where the sender occasionally inserts a CE mark itself (or reordering or loss), and checks that the receiver feeds it back faithfully [I-D.moncaster-tcpm-rcv-cheat]. This alternative requires no standardization and consumes no header bits or codepoints, as well as releasing the ECT(1) codepoint in the IP header and the NS flag in the TCP header for other uses.

3. Use Cases

ConEx is an experimental approach that allows a sender to relay congestion feedback provided by the receiver into the network along the forward data path. ConEx information can be used for traffic management to limit traffic proportionate to the actual congestion being caused, rather than limiting traffic based on rate or volume [RFC6789]. A ConEx sender uses selective acknowledgements (SACK) [RFC2018] for accurate feedback of loss signals, but currently TCP offers no equivalent accurate feedback for ECN.

DCTCP offers very low and predictable queuing delay. DCTCP changes the reaction to congestion of a TCP sender and additionally requires switches/routers to have ECN enabled and configured with a low step threshold and no signal smoothing, so it is currently only used in private networks, e.g. internal to data centers. DCTCP was released in Microsoft Windows 8, and implementations exist for Linux and FreeBSD. To retrieve sufficient congestion information, the different DCTCP implementations use a proprietary ECN feedback protocol, but they omit capability negotiation. Moreover, the feedback protocol proposed in [Ali10] only works if there are no losses at all, and otherwise it gets very confused (see Appendix A). Therefore, if a generic more accurate ECN feedback scheme were available, it would solve two problems for DCTCP: i) need for a consistent variant of DCTCP to be deployed network-wide and ii) inability to cope with ACK loss.

The following scenarios should briefly show where accurate ECN feedback is needed or adds value:

A sender with standardised TCP congestion control that supports ConEx:

In this case the ConEx mechanism uses the extra information per RTT to re-echo the precise congestion information, but the congestion control algorithm still ignores multiple marks per RTT [RFC5681].

A sender using DCTCP congestion control without ConEx:

The congestion control algorithm uses the extra info per RTT to perform its decrease depending on the number of congestion marks.

A sender using DCTCP congestion control and supporting ConEx:

Both the congestion control algorithm and ConEx use the more accurate ECN feedback mechanism.

As-yet-unspecified sender mechanisms:

The above are two examples of more general interest in sender mechanisms that respond to the extent of congestion feedback, not just its existence. It will greatly simplify incremental deployment if the sender can unilaterally deploy new behaviours, and rely on the presence of generic receivers that have already implemented more accurate feedback.

A RFC5681 TCP sender without ConEx:

No accurate feedback is necessary here. The congestion control algorithm still reacts to only one signal per RTT. But it is best to feed back all the information the receiver gets, whether the sender uses it or not -- at least as long as overhead is low or zero.

Using CE for checking integrity:

If a more accurate ECN feedback scheme feeds all occurrences of CE marks back, a sender could perform integrity checking by occasionally injecting CE marks itself. Specifically, a sender can send packets which it randomly marks with CE (at low frequency), then check if feedback is received for these packets. The congestion notification feedback for these self-injected markings, would not require a congestion control reaction [I-D.moncaster-tcpm-rcv-cheat].

4. Requirements

The requirements of the accurate ECN feedback protocol are to have fairly accurate (not necessarily perfect), timely and protected signaling. This leads to the following requirements, which MUST be discussed for any proposed more accurate ECN feedback scheme:

Resilience

The ECN feedback signal is carried within the ACK. Pure TCP ACKs can get lost without recovery (not just due to congestion, but also due to deliberate ACK thinning). Moreover, delayed ACKs are commonly used with TCP. Typically, an ACK is triggered after two data segments (or more e.g., due to receive segment coalescing, ACK compression, ACK congestion control [RFC5690] or other phenomena). In a high congestion situation where most of the packets are marked with CE, an accurate feedback mechanism should still be able to signal sufficient congestion information. Thus the accurate ECN feedback extension has to take delayed ACKs and ACK loss into account. Also, a more accurate feedback protocol should still work if delayed ACKs covered more than two packets.

Timeliness

A CE mark can be induced by a network node on the transmission path and is then echoed by the receiver in the TCP ACK. Thus when this information arrives at the sender, it is naturally already about one RTT old. With a sufficient ACK rate a further delay of a small number of packets can be tolerated. However, this information will become stale with large delays, given the dynamic nature of networks. TCP congestion control (which itself partly introduces these dynamics) operates on a time scale of one RTT. Thus, to be timely, congestion feedback information should be delivered within about one RTT.

Integrity

It should be possible to assure the integrity of the feedback in a more accurate ECN feedback scheme, at least as well as the ECN Nonce. Alternatively, it should at least be possible to give strong incentives for the receiver and network nodes to cooperate honestly.

Given there are known problems with the ECN nonce (as identified above), this document only requires that the integrity of the more accurate ECN feedback can be assured as an inherent part of the new more accurate ECN feedback

protocol; it does not require that the ECN Nonce mechanism is employed to achieve this. Indeed, if integrity could be provided else-wise, a more accurate ECN feedback protocol might re-purpose the nonce sum (NS) flag in the TCP header.

If the more accurate ECN feedback scheme provides sufficient information, the integrity check could e.g. be performed by deterministically setting the CE in the sender and monitoring the respective feedback (similar to ECT(1) and the ECN Nonce sum). Whether a sender should enforce when it detects wrong feedback information, and what kind of enforcement it should apply, are policy issues that need not be specified as part of more accurate ECN feedback scheme.

Accuracy

Classic ECN feeds back one congestion notification per RTT, which is sufficient for classic TCP congestion control which reduces the sending rate at most once per RTT. Thus the more accurate ECN feedback scheme should ensure that, if a congestion episode occurs, at least one congestion notification is echoed and received per RTT as classic ECN would do. Of course, the goal of a more accurate ECN extension is to reconstruct the number of CE markings more accurately. In the best case the new scheme should even allow reconstruction of the exact number of payload bytes that a CE marked packet was carrying. However, it is accepted that it may be too complex for a sender to get the exact number of congestion markings or marked bytes in all situations. Ideally, the feedback scheme should preserve the order in which any (of the four) ECN signals were received. And, ideally, it would even be possible for the sender to determine which of the packets covered by one delayed ACK were congestion marked, e.g. if the flow consists of packets of different sizes, or to allow for future protocols where the order of the markings may be important.

In the best case, a sender that sees more accurate ECN feedback information would be able to reconstruct the occurrence of any of the four code points (non-ECT, CE, ECT(0), ECT(1)). However, assuming the sender marks all data packets as ECN-capable and uses the default setting of ECT(0), solely feeding back the occurrence of CE and ECT(1) might be sufficient. Thus a more accurate ECN feedback scheme should at least provide information on these two signals, CE and ECT(1).

If a more accurate ECN scheme can reliably deliver feedback in most but not all circumstances, ideally the scheme should

at least not introduce bias. In other words, undetected loss of some ACKs should be as likely to increase as decrease the sender's estimate of the probability of ECN marking.

Complexity

Implementation should be as simple as possible and only a minimum of additional state information should be needed. This will enable more accurate ECN feedback to be used as the default feedback mechanism, even if only one ECN feedback signal per RTT is needed. Furthermore, the receiver should not make assumptions about the mechanism that was used to set the markings nor about any interpretation or reaction to the congestion signal. The receiver only needs to faithfully reflect congestion information back to the sender.

Overhead

A more accurate ECN feedback signal should limit the additional network load, because ECN feedback is ultimately not critical information (in the worst case, loss will still be available as a congestion signal of last resort). As feedback information has to be provided frequently and in a timely fashion, potentially all or a large fraction of TCP acknowledgments might carry this information. Ideally, no additional segments should be exchanged compared to an RFC3168 TCP session, and the overhead in each segment should be minimized.

Backward and forward compatibility

Given more accurate ECN feedback will involve a change to the TCP protocol, it should to be negotiated between the two TCP endpoints. If either end does not support the more accurate feedback, they should both be able to fall-back to classic ECN feedback.

A more accurate ECN feedback extension should aim to be able to traverse most existing middleboxes. Further, a feedback mechanism should provide a method to fall-back to classic ECN signaling if the new signal is suppressed by certain middleboxes.

In order to avoid a fork in the TCP protocol specifications, if experiments with the new ECN feedback protocol are successful, it is intended to eventually update RFC3168 for any TCP/ECN sender, not just for ConEx or DCTCP senders. Then future senders will be able to unilaterally deploy new behaviours that exploit the existence of more accurate ECN feedback in receivers (forward compatibility). Conversely, even if another sender only needs one ECN feedback signal per

RTT, it should be able to use more accurate ECN feedback, and simply ignore the excess information.

5. Design Approaches

All approaches presented below (and proposed so far) are able to provide accurate ECN feedback information as long as no ACK loss occurs and the congestion rate is reasonable. In case of a high ACK loss rate or very high congestion (CE marking) rate, the proposed schemes have different resilience characteristics depending on the number of bits used for the encoding. While classic ECN provides reliable (but inaccurate) feedback of a maximum of one congestion signal per RTT, the proposed schemes do not implement an explicit acknowledgement mechanism for the feedback (as e.g. the ECE / CWR exchange of [RFC3168]).

5.1. Re-Definition of ECN/NS Header Bits

Schemes in this category can additionally use the NS bit for capability negotiation during the TCP handshake exchange. Thus a more accurate ECN could be negotiated without changing the classic ECN negotiation and thus being backwards compatible.

Schemes in this category can simply re-define the ECN header flags, ECE and CWR, to encode the occurrence of a CE marking at the receiver. This approach provides very limited resilience against loss of ACK, particularly pure ACKs (no payload and therefore delivered unreliably).

A couple of schemes have been proposed so far:

- o A naive one-bit scheme that sends one ECE for each CE received could use CWR to increase robustness against ACK loss by introducing redundant information on the next ACK, but this is still highly vulnerable to ACK loss.
- o The scheme defined for DCTCP [Ali10], which toggles the ECE feedback on an immediate ACK whenever the CE marking changes, and otherwise feeds back delayed ACKs with the ECE value unchanged. Appendix A demonstrates that this scheme is still highly ambiguous to the sender if the ACKs are pure ACKs, and if some may have been lost.

Alternatively, the receiver uses the three ECN/NS header flags, ECE, CWR and NS to represent a counter that signals the accumulated number of CE markings it has received. Resilience against loss is better than the flag-based schemes, but still not ideal.

A couple of coding schemes have been proposed so far in this category:

- o A 3-bit counter scheme continuously feeds back the three least significant bits of a CE counter;
- o A scheme that defines a standardised lookup table to map the 8 codepoints onto either a CE counter or an ECT(1) counter.

These proposed schemes provide accumulated information on ECN-CE marking feedback, similar to the number of acknowledged bytes in the TCP header. Due to the limited number of bits the ECN feedback information will wrap much more often than the acknowledgement field. Thus feedback information could be lost due to a relatively small sequence of pure-ACK losses. Resilience could be increased by introducing redundancy, e.g. send each counter increase two or more times. Of course any of these additional mechanisms will increase the complexity. If the congestion rate is greater than the ACK rate (multiplied by the number of congestion marks that can be signaled per ACK), the congestion information cannot correctly be fed back. Covering the worst case where every packet is CE marked can potentially be realized by dynamically adapting the ACK rate and redundancy. This again increases complexity and perhaps the signaling overhead as well. Schemes that do not re-purpose the ECN NS bit, could still support the ECN Nonce.

5.2. Using Other Header Bits

As seen in Figure 1, there are currently three unused flags in the TCP header. The proposed 3-bit counter or codepoint schemes could be extended by one or more bits to add higher resilience against ACK loss. The relative gain would be exponentially higher resilience against ACK loss, while the respective drawbacks would remain identical.

Alternatively, the receiver could use bits in the Urgent Pointer field to signal more bits of its congestion signal counter, but only whenever it does not set the Urgent Flag. As this is often the case, resilience could be increased without additional header overhead.

Any proposal to use such bits would need to check the likelihood that some middleboxes might discard or 'normalize' the currently unused flag bits or a non-zero Urgent Pointer when the Urgent Flag is cleared.

5.3. Using a TCP Option

Alternatively, a new TCP option could be introduced, to help maintain the accuracy and integrity of ECN feedback between receiver and sender. Such an option could provide higher resilience and even more information. E.g. ECN for RTP/UDP [RFC6679] explicitly provides the number of ECT(0), ECT(1), CE, non-ECT marked and lost packets, and SCTP counts the number of ECN marks [I-D.stewart-tsvwg-sctpecn] between CWR chunks. However, deploying new TCP options has its own challenges. Moreover, to actually achieve high resilience, this option would need to be carried by most or all ACKs. Thus this approach would introduce considerable signaling overhead even though ECN feedback is not extremely critical information (in the worst case, loss will still be available to provide a strong congestion feedback signal). Whatever, such a TCP option could be used in addition to a more accurate ECN feedback scheme in the TCP header or in addition to classic ECN, only when needed and when space is available.

6. Acknowledgements

Thanks to Gorrry Fairhurst for ideas on CE-based integrity checking and to Mohammad Alizadeh for suggesting the need to avoid bias. Moverover, thanks to Michael Welzl and Michael Scharf for their feedback.

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

Given ECN feedback is used as input for congestion control, the respective algorithm would not react appropriately if ECN feedback were lost and the resilience mechanism to recover it was inadequate. This resilience requirement is articulated in Section 4. However, it should be noted that ECN feedback is not the last resort against congestion collapse, because if there is insufficient response to ECN, loss will ensue, and TCP will still react appropriately to loss.

A receiver could suppress ECN feedback information leading to its connections consuming excess sender or network resources. This problem is similar to that seen with the classic ECN feedback scheme and should be addressed by integrity checking as required in Section 4.

9. References

9.1. 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.

9.2. Informative References

- [Ali10] Alizadeh, M., Greenberg, A., Maltz, D., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and M. Sridharan, "Data Center TCP (DCTCP)", ACM SIGCOMM CCR 40(4)63-74, October 2010, <<http://portal.acm.org/citation.cfm?id=1851192>>.
- [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-02 (work in progress), November 2007.
- [I-D.stewart-tsvwg-sctpecn] Stewart, R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", draft-stewart-tsvwg-sctpecn-05 (work in progress), January 2014.
- [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.
- [RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, February 2010.
- [RFC6679] Westerlund, M., Johansson, I., Perkins, C., O'Hanlon, P., and K. Carlberg, "Explicit Congestion Notification (ECN) for RTP over UDP", RFC 6679, August 2012.

[RFC6789] Briscoe, B., Woundy, R., and A. Cooper, "Congestion Exposure (ConEx) Concepts and Use Cases", RFC 6789, December 2012.

Appendix A. Ambiguity of the More Accurate ECN Feedback in DCTCP

As defined in [Ali10], a DCTCP receiver feeds back ECE=0 on delayed ACKs as long as CE remains 0, and also immediately sends an ACK with ECE=0 when CE transitions to 1. Similarly, it continually feeds back ECE=1 on delayed ACKs while CE remains 1 and immediately feeds back ECE=1 when CE transitions to 0. A sender can unambiguously decode this scheme if there is never any ACK loss, and the sender assumes there will never be any ACK loss.

The following two examples show that the feedback sequence becomes highly ambiguous to the sender, if either of these conditions is broken. Below, '0' will represent ECE=0, '1' will represent ECE=1 and '.' will represent a gap of one segment between delayed ACKs. Now imagine that the sender receives the following sequence of feedback on 3 pure ACKs:

0.0.0

When the receiver sent this sequence it could have been any of the following four sequences:

- a. 0.0.0 (0 x CE)
- b. 010.0 (1 x CE)
- c. 0.010 (1 x CE)
- d. 01010 (2 x CE)

where any of the 1s represent a possible pure ACK carrying ECE feedback that could have been lost. If the sender guesses (a), it might be correct, or it might miss 1 or 2 congestion marks over 5 packets. Therefore, when confronted with this simple sequence (that is not contrived), a sender can guess that congestion might have been 0%, 20% or 40%, but it doesn't know which.

Sequences with a longer gap (e.g. 0...0.0) become far more ambiguous. It helps a little if the sender knows the distance the receiver uses between delayed ACKs, and it helps a lot if the distance is 1, i.e. no delayed ACKs, but even then there will still be ambiguity whenever there are pure ACK losses.

Authors' Addresses

Mirja Kuehlewind (editor)
University of Stuttgart
Pfaffenwaldring 47
Stuttgart 70569
Germany

Email: 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

Bob Briscoe
BT
B54/77, Adastral Park
Martlesham Heath
Ipswich IP5 3RE
UK

Phone: +44 1473 645196
Email: bob.briscoe@bt.com
URI: <http://bobbbriscoe.net/>

TCPM Working Group
Internet-Draft
Obsoletes: 2861 (if approved)
Updates: 5681 (if approved)
Intended status: Standards Track
Expires: August 8, 2014

G. Fairhurst
A. Sathiaselan
R. Secchi
University of Aberdeen
February 4, 2014

Updating TCP to support Rate-Limited Traffic
draft-ietf-tcpm-newcwv-05

Abstract

This document proposes an update to RFC 5681 to address issues that arise when TCP is used to support traffic that exhibits periods where the sending rate is limited by the application rather than the congestion window. It updates TCP to allow a TCP sender to restart quickly following either an idle or rate-limited interval. This method is expected to benefit applications that send rate-limited traffic using TCP, while also providing an appropriate response if congestion is experienced.

It also evaluates the Experimental specification of TCP Congestion Window Validation, CWV, defined in RFC 2861, and concludes that RFC 2861 sought to address important issues, but failed to deliver a widely used solution. This document therefore recommends that the status of RFC 2861 is moved from Experimental to Historic, and that it is replaced by the current specification.

NOTE: The standards status of this WG document is under review for consideration as either Experimental (EXP) or Proposed Standard (PS). This decision will be made later as the document is finalised.

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 8, 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	3
2. Reviewing experience with TCP-CWV	4
3. Terminology	4
4. An updated TCP response to idle and application-limited periods	5
4.1. A method for preserving cwnd during the idle and application-limited periods.	6
4.2. Initialisation	7
4.3. The nonvalidated phase	7
4.4. TCP congestion control during the nonvalidated phase	7
4.4.1. Response to congestion in the nonvalidated phase	9
4.4.2. Sender burst control during the nonvalidated phase	10
4.4.3. Adjustment at the end of the nonvalidated phase	10
4.5. Examples of Implementation	11
4.5.1. Implementing pipeACK	11
4.5.2. Implementing detection of the cwnd-limited condition	12
5. Determining a safe period to preserve cwnd	12
6. Security Considerations	13
7. IANA Considerations	14
8. Acknowledgments	14
9. Author Notes	14
9.1. Other related work	14
9.2. Revision notes	16
10. References	18
10.1. Normative References	18
10.2. Informative References	19
Authors' Addresses	20

1. Introduction

TCP is used to support a range of application behaviours. The TCP congestion window (cwnd) controls the number of unacknowledged packets/bytes that a TCP flow may have in the network at any time, a value known as the FlightSize [RFC5681]. A bulk application will always have data available to transmit. The rate at which it sends is therefore limited by the maximum permitted by the receiver advertised window and the sender congestion window (cwnd). In contrast, a rate-limited application will experience periods when the sender is either idle or is unable to send at the maximum rate permitted by the cwnd. This latter case is called rate-limited. The focus of this document is on the operation of TCP in such an idle or rate-limited case.

Standard TCP [RFC5681] requires the cwnd to be reset to the restart window (RW) when an application becomes idle. [RFC2861] noted that this TCP behaviour was not always observed in current implementations. Recent experiments [Bis08] confirm this to still be the case.

Standard TCP does not impose additional restrictions on the growth of the cwnd when a TCP sender is rate-limited. A rate-limited sender may therefore grow a cwnd far beyond that corresponding to the current transmit rate, resulting in a value that does not reflect current information about the state of the network path the flow is using. Use of such an invalid cwnd may result in reduced application performance and/or could significantly contribute to network congestion.

[RFC2861] proposed a solution to these issues in an experimental method known as Congestion Window Validation (CWV). CWV was intended to help reduce cases where TCP accumulated an invalid cwnd. The use and drawbacks of using the CWV algorithm in RFC 2861 with an application are discussed in Section 2.

Section 3 defines relevant terminology.

Section 4 specifies an alternative to CWV that seeks to address the same issues, but does this in a way that is expected to mitigate the impact on an application that varies its sending rate. The method described applies to both a rate-limited and an idle condition. Section 5 describes the rationale for selecting the safe period to preserve the cwnd.

2. Reviewing experience with TCP-CWV

RFC 2861 described a simple modification to the TCP congestion control algorithm that decayed the cwnd after the transition to a "sufficiently-long" idle period. This used the slow-start threshold (ssthresh) to save information about the previous value of the congestion window. The approach relaxed the standard TCP behaviour [RFC5681] for an idle session, intended to improve application performance. CWV also modified the behaviour for a rate-limited session where a sender transmitted at a rate less than allowed by cwnd.

RFC 2861 has been implemented in some mainstream operating systems as the default behaviour [Bis08]. Analysis (e.g. [Bis10] [Fail2]) has shown that a TCP sender using CWV is able to use available capacity on a shared path after an idle period. This can benefit some applications, especially over long delay paths, when compared to the slow-start restart specified by standard TCP. However, CWV would only benefit an application if the idle period were less than several Retransmission Time Out (RTO) intervals [RFC6298], since the behaviour would otherwise be the same as for standard TCP, which resets the cwnd to the RTCP Restart Window (RW) after this period.

Experience with RFC 2861 suggests that although the CWV method benefited the network in a rate-limited scenario (reducing the probability of network congestion), the behaviour was too conservative for many common rate-limited applications. This mechanism did not therefore offer the desirable increase in application performance for rate-limited applications and it is unclear whether applications actually use this mechanism in the general Internet.

It is therefore concluded that CWV, as defined in RFC2681, was often a poor solution for many rate-limited applications. It had the correct motivation, but had the wrong approach to solving this problem.

3. 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 document assumes familiarity with the terminology of TCP congestion control [RFC5681].

The following new terminology is introduced:

cwnd-limited: A TCP flow that has sent the maximum number of segments permitted by the cwnd, where the application utilises the allowed sending rate (see section 4.5.2).

pipeACK sample: A measure of the volume of data acknowledged by the network within an RTT.

pipeACK variable: A variable that measures the available capacity using the set of pipeACK samples.

pipeACK Sampling Period: The maximum period that a measured pipeACK sample may influence the pipeACK variable.

Non-validated phase: The phase where the cwnd reflects a previous measurement of the available path capacity.

Non-validated period, NVP: The maximum period for which cwnd is preserved in the non-validated phase.

Rate-limited: A TCP flow that does not consume more than one half of cwnd, and hence operates in the non-validated phase.

Validated phase: The phase where the cwnd reflects a current estimate of the available path capacity.

4. An updated TCP response to idle and application-limited periods

This section proposes an update to the TCP congestion control behaviour during a rate-limited period. The new method permits a TCP sender to preserve the cwnd when an application becomes idle or when the sender is unable to send at the maximum rate permitted by the cwnd (the non-validated period, NVP, see section 5). The period where actual usage is less than allowed by cwnd, is named as the non-validated phase. This method allows an application to resume transmission at a previous rate without incurring the delay of slow-start. However, if the TCP sender experiences congestion using the preserved cwnd, it is required to immediately reset the cwnd to an appropriate value specified by the method. If a sender does not take advantage of the preserved cwnd within the NVP, the value of cwnd is reduced, ensuring the value better reflects the capacity that was recently actually used.

It is expected that this update will satisfy the requirements of many rate-limited applications and at the same time provide an appropriate method for use in the Internet. It also reduces the incentive for an application to send data simply to keep transport congestion state. (This is sometimes known as "padding").

The new method does not differentiate between times when the sender has become idle or rate-limited. This is partly a response to recognition that some applications wish to transmit at a rate less than allowed by the sender cwnd, and that it can be hard to make a distinction between rate-limited and idle behaviour. This is expected to encourage applications and TCP stacks to use standards-based congestion control methods. It may also encourage the use of long-lived connections where this offers benefit (such as persistent http).

The method is specified in following subsections.

4.1. A method for preserving cwnd during the idle and application-limited periods.

[RFC5681] defines a variable, FlightSize, that indicates the amount of outstanding data in the network. This is assumed to be equal to the value of Pipe calculated based on the pipe algorithm [RFC3517]. In RFC5681 this value is used during loss recovery, whereas in this method a new variable "pipeACK" is introduced to measure the acknowledged size of the pipe, which is used to determine if the sender has validated the cwnd.

A sender determines a pipeACK sample by measuring the volume of data that was acknowledged by the network over the period of a measured Round Trip Time (RTT). Using the variables defined in [RFC3517], a value could be measured by caching the value of HighACK and after one RTT measuring the difference between the cached HighACK value and the current HighACK value. Other equivalent methods may be used.

A sender is not required to continuously update the pipeACK variable after each received ACK, but SHOULD perform a pipeACK sample at least once per RTT when it has sent unacknowledged segments.

The pipeACK variable MAY consider multiple pipeACK samples over the pipeACK Sampling Period. The value of the pipeACK variable MUST NOT exceed the maximum (highest value) within the sampling period. This specification defines the pipeACK Sampling Period as $\text{Max}(3 \cdot \text{RTT}, 1 \text{ second})$. This period enables a sender to compensate for large fluctuations in the sending rate, where there may be pauses in transmission, and allows the pipeACK variable to reflect the largest recently measured pipeACK sample.

When no measurements are available, the pipeACK variable is set to the "undefined value". This value is used to inhibit entering the nonvalidated phase until the first new measurement of a pipeACK sample.

The method RECOMMENDS that the TCP SACK option [RFC3517] is enabled. This allows the sender to more accurately determine the number of missing bytes during the loss recovery phase, and using this method will result in a higher cwnd following loss.

4.2. Initialisation

A sender starts a TCP connection in the Validated phase and initialises the pipeACK variable to the "undefined" value. This value inhibits use of the value in cwnd calculations.

4.3. The nonvalidated phase

The updated method creates a new TCP sender phase that captures whether the cwnd reflects a validated or non-validated value. The phases are defined as:

- o Validated phase: $\text{pipeACK} \geq (1/2) * \text{cwnd}$, or pipeACK is undefined. This is the normal phase, where cwnd is expected to be an approximate indication of the capacity currently available along the network path, and the standard methods are used to increase cwnd (currently [RFC5681]). The rule for transitioning to the non-validated phase is specified in section 4.4.
- o Non-validated phase: $\text{pipeACK} < (1/2) * \text{cwnd}$. This is the phase where the cwnd has a value based on a previous measurement of the available capacity, and the usage of this capacity has not been validated in the pipeACK Sampling Period. That is, when it is not known whether the cwnd reflects the currently available capacity along the network path. The mechanisms to be used in this phase seek to determine a safe value for cwnd and an appropriate reaction to congestion. These mechanisms are specified in section 4.4.

The value 1/2 was selected to reduce the effects of variations in the pipeACK variable, and to allow the sender some flexibility in when it sends data.

4.4. TCP congestion control during the nonvalidated phase

A TCP sender MUST enter the non-validated phase when the pipeACK is less than $(1/2) * \text{cwnd}$.

A TCP sender that enters the non-validated phase will preserve the cwnd (i.e., this neither grows nor reduces while the sender remains in this phase). If the sender receives an indication of congestion (loss or Explicit Congestion Notification, ECN, mark [RFC3168]) it uses the method described below. The phase is concluded after a

fixed period of time (the NVP, as explained in section 4.4.3) or when the sender transmits sufficient data so that $\text{pipeACK} > (1/2) * \text{cwnd}$ (i.e. it is no longer rate-limited).

The behaviour in the non-validated phase is specified as:

- o A sender determines whether to increase the cwnd based upon whether it is cwnd-limited (see section 4.5.2):
 - o
 - * A sender that is cwnd-limited MAY use the standard TCP method to increase cwnd (i.e. a TCP sender that fully utilises the cwnd is permitted to increase cwnd each received ACK using standard methods).
 - * A sender that is not cwnd-limited MUST NOT increase the cwnd when ACK packets are received in this phase.
- o If the sender receives an indication of congestion while in the non-validated phase (i.e., detects loss, or an ECN mark), the sender MUST exit the non-validated phase (reducing the cwnd as defined in section 4.4.1).
- o If the Retransmission Time Out (RTO) expires while in the non-validated phase, the sender MUST exit the non-validated phase. It then resumes using the Standard TCP RTO mechanism [RFC5681]. (The resulting reduction of cwnd described in section 4.4.3 is appropriate, since any accumulated path history is considered unreliable).
- o A sender with a pipeACK variable greater than $(1/2) * \text{cwnd}$ SHOULD enter the validated phase. (A rate-limited sender will not normally be impacted by whether it is in a validated or non-validated phase, since it will normally not consume the entire cwnd. However a change to the validated phase will release the sender from constraints on the growth of cwnd, and restore the use of the standard congestion response.)

The cwnd-limited behaviour may be triggered during a transient condition that occurs when a sender is in the non-validated phase and receives an ACK that acknowledges received data, the cwnd was fully utilised, and more data is awaiting transmission than may be sent with the current cwnd. The sender is then allowed to use the standard method to increase the cwnd. (Note, if the sender succeeds in sending these new segments, the updated cwnd and pipeACK variables will eventually result in a transition to the validated phase.)

4.4.1. Response to congestion in the nonvalidated phase

Reception of congestion feedback while in the non-validated phase is interpreted as an indication that it was inappropriate for the sender to use the preserved cwnd. The sender is therefore required to quickly reduce the rate to avoid further congestion. Since the cwnd does not have a validated value, a new cwnd value must be selected based on the utilised rate.

A sender that detects a packet-drop, or receives an indication of an ECN marked packet, MUST record the current FlightSize in the variable LossFlightSize and MUST calculate a safe cwnd for loss recovery using the method below:

$$\text{cwnd} = (\text{Max}(\text{pipeACK}, \text{LossFlightSize})) / 2.$$

If there is a valid pipeACK value, the new cwnd is adjusted to reflect that a nonvalidated cwnd may be larger than the actual FlightSize, or recently used FlightSize (recorded in pipeACK). The updated cwnd therefore prevents overshoot by a sender significantly increasing its transmission rate during the recovery period.

At the end of the recovery phase, the TCP sender MUST reset the cwnd using the method below:

$$\text{cwnd} = (\text{Max}(\text{pipeACK}, \text{LossFlightSize}) - R) / 2.$$

Where, R is the volume of data that was retransmitted during the recovery phase. This follows the method proposed for Jump Start [Liu07]. The inclusion of the term R makes an adjustment more conservative than standard TCP. (This is required, since the sender may have sent more segments than a standard TCP sender would have done. The additional reduction is beneficial when the LossFlightSize significantly overshoots the available path capacity incurring significant loss, for instance an intense traffic burst following a non-validated period.)

If the sender implements a method that allows it to identify the number of ECN-marked segments within a window that were observed by the receiver, the sender SHOULD use the method above, further reducing R by the number of marked segments.

The sender MUST also re-initialise the pipeACK variable to the "undefined" value. This ensures that standard TCP methods are used immediately after completing loss recovery until a new pipeACK value can be determined.

ssthresh is adjusted using the standard TCP method.

4.4.2. Sender burst control during the nonvalidated phase

TCP congestion control allows a sender to accumulate a cwnd that would allow it to send a burst of segments with a total size up to the difference between the FlightsSize and cwnd. Such bursts can impact other flows that share a network bottleneck and/or may induce congestion when buffering is limited.

Various methods have been proposed to control the sender burstiness [Hug01], [All05]. For example, TCP can limit the number of new segments it sends per received ACK. This is effective when a flow of ACKs is received, but can not be used to control a sender that has not send appreciable data in the previous RTT [All05].

This document recommends using a method to avoid line-rate bursts after an idle or rate-limited period when there is less reliable information about the capacity of the network path: A TCP sender in the non-validated phase SHOULD control the maximum burst size, e.g. using a rate-based pacing algorithm in which a sender paces out the cwnd over its estimate of the RTT, or some other method, to prevent many segments being transmitted contiguously at line-rate. The most appropriate method(s) to implement pacing depend on the design of the TCP/IP stack, speed of interface and whether hardware support (such as TCP Segment Offload, TSO) is used. The present document does not recommend any specific method.

4.4.3. Adjustment at the end of the nonvalidated phase

An application that remains in the non-validated phase for a period greater than the NVP is required to adjust its congestion control state. If the sender exits the non-validated phase after this period, it MUST update the ssthresh:

$$\text{ssthresh} = \max(\text{ssthresh}, 3 * \text{cwnd} / 4).$$

(This adjustment of ssthresh ensures that the sender records that it has safely sustained the present rate. The change is beneficial to rate-limited flows that encounter occasional congestion, and could otherwise suffer an unwanted additional delay in recovering the sending rate.)

The sender MUST then update cwnd to be not greater than:

$$\text{cwnd} = \max((1/2) * \text{cwnd}, \text{IW}).$$

Where IW is the appropriate TCP initial window, used by the TCP sender (e.g. [RFC5681]).

(This adjustment ensures that sender responds conservatively at the end of the non-validated phase by reducing the cwnd to better reflect the current rate of the sender. The cwnd update does not take into account FlightSize or pipeACK value because these values only reflect historical data and do not reflect the current sending rate.)

4.5. Examples of Implementation

This section provides informative examples of implementation methods. Implementations may choose to use other methods that comply with the normative requirements.

4.5.1. Implementing pipeACK

A pipeACK sample may be measured once each RTT. This reduces the sender processing burden for calculating after each acknowledgement and also reduces storage requirements at the sender.

Since application behaviour can be bursty using CWV, it may be desirable to implement a maximum filter to accumulate the measured values so that the pipeACK variable records the largest pipeACK sample within the pipeACK Sampling Period. One simple way to implement this is to divide the pipeACK Sampling Period into several (e.g. 5) equal length measurement periods. The sender then records the start time for each measurement period and the highest measured pipeACK sample. At the end of the measurement period, any measurement(s) that are older than the pipeACK Sampling Period are discarded. The pipeACK variable is then assigned the largest of the set of the highest measured values.

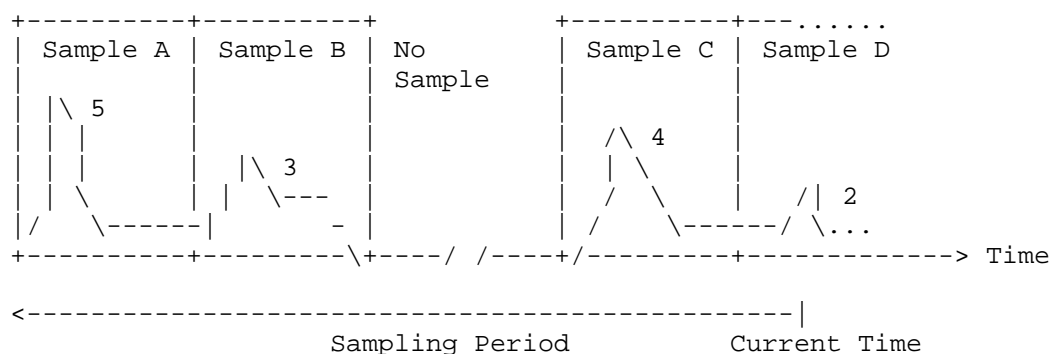


Figure 1: Example of measuring pipeACK samples

Figure 1 shows an example of how measurement samples may be collected. At the time represented by the figure new samples are being accumulated into sample D. Three previous samples also fall within the pipeACK Sampling Period: A, B, and C. There was also a period of inactivity between samples B and C during which no measurements were taken. The current value of the pipeACK variable will be 5, the maximum across all samples.

After one further measurement period, Sample A will be discarded, since it then is older than the pipeACK Sampling Period and the pipeACK variable will be recalculated, Its value will be the larger of Sample C or the final value accumulated in Sample D.

Note that the pipeACK Sampling Period and the NVP period do not necessarily require a new timer to be implemented. An alternative is to record a timestamp when the sender enters the NVP. Each time a sender transmits a new segment, this timestamp may be used to determine if the NVP period has expired. If the period expires, the sender may take into account how many units of the NVP period have passed and make one reduction (as defined in section 4.4.3) for each NVP period.

4.5.2. Implementing detection of the cwnd-limited condition

A method is required to detect the cwnd-limited condition (see section 4.4). This is used to detect a condition where a sender in the non-validated phase receives an ACK, but the size of cwnd prevents sending more new data.

In simple terms this condition is true only when the TCP sender's FlightSize is equal to or larger than the cwnd. However, an implementation must consider other constraints on the way in which cwnd variable is used, for instance the need to support methods such as the Nagle Algorithm and TCP Segment Offload (TSO). This can result in a sender becoming cwnd-limited when the cwnd is nearly, rather than completely, equal to the FlightSize.

5. Determining a safe period to preserve cwnd

This section documents the rationale for selecting the maximum period that cwnd may be preserved, known as the non-validated period, NVP.

Limiting the period that cwnd may be preserved avoids undesirable side effects that would result if the cwnd were to be kept unnecessarily high for an arbitrary long period, which was a part of the problem that CWV originally attempted to address. The period a sender may safely preserve the cwnd, is a function of the period that

a network path is expected to sustain the capacity reflected by cwnd. There is no ideal choice for this time.

A period of five minutes was chosen for this NVP. This is a compromise that was larger than the idle intervals of common applications, but not sufficiently larger than the period for which the capacity of an Internet path may commonly be regarded as stable. The capacity of wired networks is usually relatively stable for periods of several minutes and that load stability increases with the capacity. This suggests that cwnd may be preserved for at least a few minutes.

There are cases where the TCP throughput exhibits significant variability over a time less than five minutes. Examples could include wireless topologies, where TCP rate variations may fluctuate on the order of a few seconds as a consequence of medium access protocol instabilities. Mobility changes may also impact TCP performance over short time scales. Senders that observe such rapid changes in the path characteristic may also experience increased congestion with the new method, however such variation would likely also impact TCP's behaviour when supporting interactive and bulk applications.

Routing algorithms may modify the network path, disrupting the RTT measurement and changing the capacity available to a TCP connection, however such changes do not often occur within a time frame of a few minutes.

The value of five minutes is therefore expected to be sufficient for most current applications. Simulation studies (e.g. [Bis11]) also suggest that for many practical applications, the performance using this value will not be significantly different to that observed using a non-standard method that does not reset the cwnd after idle.

Finally, other TCP sender mechanisms have used a 5 minute timer, and there could be simplifications in some implementations by reusing the same interval. TCP defines a default user timeout of 5 minutes [RFC0793] i.e. how long transmitted data may remain unacknowledged before a connection is forcefully closed.

6. Security Considerations

General security considerations concerning TCP congestion control are discussed in [RFC5681]. This document describes an algorithm that updates one aspect of the congestion control procedures, and so the considerations described in RFC 5681 also apply to this algorithm.

7. IANA Considerations

There are no IANA considerations.

8. Acknowledgments

The authors acknowledge the contributions of Dr I Biswas, Mr Ziaul Hossain in supporting the evaluation of CWV and for their help in developing the mechanisms proposed in this draft. We also acknowledge comments received from the Internet Congestion Control Research Group, in particular Yuchung Cheng, Mirja Kuehlewind, and Joe Touch. This work was part-funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700).

9. Author Notes

RFC-Editor note: please remove this section prior to publication.

9.1. Other related work

RFC-Editor note: please remove this section prior to publication.

There are several issues to be discussed more widely:

- o There are potential interactions with the Experimental update in [RFC6928] that raises the TCP initial Window to ten segments, do these cases need to be elaborated?

This relates to the Experimental specification for increasing the TCP IW defined in RFC 6928.

The two methods have different functions and different response to loss/congestion.

RFC 6928 proposes an experimental update to TCP that would increase the IW to ten segments. This would allow faster opening of the cwnd, and also a large (same size) restart window. This approach is based on the assumption that many forward paths can sustain bursts of up to ten segments without (appreciable) loss. Such a significant increase in cwnd must be matched with an equally large reduction of cwnd if loss/congestion is detected, and such a congestion indication is likely to require future use of IW=10 to be disabled for this path for some time. This guards against the unwanted behaviour

of a series of short flows continuously flooding a network path without network congestion feedback.

In contrast, this document proposes an update with a rationale that relies on recent previous path history to select an appropriate cwnd after restart.

The behaviour differs in three ways:

1) For applications that send little initially, new-cwv may constrain more than RFC 6928, but would not require the connection to reset any path information when a restart incurred loss. In contrast, new-cwv would allow the TCP connection to preserve the cached cwnd, any loss, would impact cwnd, but not impact other flows.

2) For applications that utilise more capacity than provided by a cwnd of 10 segments, this method would permit a larger restart window compared to a restart using the method in RFC 6928. This is justified by the recent path history.

3) new-CWV is intended to also be used for rate-limited applications, where the application sends, but does not seek to fully utilise the cwnd. In this case, new-cwv constrains the cwnd to that justified by the recent path history. The performance trade-offs are hence different, and it would be possible to enable new-cwv when also using the method in RFC 6928, and yield benefits.

o There is potential overlap with the Laminar proposal (draft-mathis-tcpm-tcp-laminar)

The current draft was intended as a standards-track update to TCP, rather than a new transport variant. At least, it would be good to understand how the two interact and whether there is a possibility of a single method.

o There is potential performance loss in loss of a short burst (off list with M Allman)

A sender can transmit several segments then become idle. If the first segments are all ACK'ed the ssthresh collapses to a small value (no new data is sent by the idle sender). Loss of the later data results in congestion (e.g. maybe a RED drop or

some other cause, rather than the maximum rate of this flow). When the sender performs loss recovery it may have an appreciable pipeACK and cwnd, but a very low FlightSize - the Standard algorithm results in an unusually low cwnd $((1/2) * \text{FlightSize})$.

A constant rate flow would have maintained a FlightSize appropriate to pipeACK (cwnd if it is a bulk flow).

This could be fixed by adding a new state variable? It could also be argued this is a corner case (e.g. loss of only the last segments would have resulted in RTO), the impact could be significant.

- o There is potential interaction with TCP Control Block Sharing(MWelzl)

An application that is non-validated can accumulate a cwnd that is larger than the actual capacity. Is this a fair value to use in TCB sharing?

We propose that TCB sharing should use the pipeACK in place of cwnd when a TCP sender is in the Nonvalidated phase. This value better reflects the capacity that the flow has utilised in the network path.

9.2. Revision notes

RFC-Editor note: please remove this section prior to publication.

Draft 03 was submitted to ICCRG to receive comments and feedback.

Draft 04 contained the first set of clarifications after feedback:

- o Changed name to application limited and used the term rate-limited in all places.
- o Added justification and many minor changes suggested on the list.
- o Added text to tie-in with more accurate ECN marking.
- o Added ref to Hug01

Draft 05 contained various updates:

- o New text to redefine how to measure the acknowledged pipe, differentiating this from the FlightSize, and hence avoiding previous issues with infrequent large bursts of data not being validated. A key point new feature is that pipeACK only triggers leaving the NVP after the size of the pipe has been acknowledged. This removed the need for hysteresis.
- o Reduction values were changed to 1/2, following analysis of suggestions from ICCRG. This also sets the "target" cwnd as twice the used rate for non-validated case.
- o Introduced a symbolic name (NVP) to denote the 5 minute period.

Draft 06 contained various updates:

- o Required reset of pipeACK after congestion.
- o Added comment on the effect of congestion after a short burst (M. Allman).
- o Correction of minor Typos.

WG draft 00 contained various updates:

- o Updated initialisation of pipeACK to maximum value.
- o Added note on intended status still to be determined.

WG draft 01 contained:

- o Added corrections from Richard Scheffenegger.
- o Raffaello Secchi added to the mechanism, based on implementation experience.
- o Removed that the requirement for the method to use TCP SACK option [RFC3517] to be enabled - Although it may be desirable to use SACK, this is not essential to the algorithm.
- o Added the notion of the sampling period to accommodate large rate variations and ensure that the method is stable. This algorithm to be validated through implementation.

WG draft 02 contained:

- o Clarified language around pipeACK variable and pipeACK sample - Feedback from Aris Angelogiannopoulos.

WG draft 03 contained:

- o Editorial corrections - Feedback from Anna Brunstrom.
- o An adjustment to the procedure at the start and end of loss recovery to align the two equations.
- o Further clarification of the "undefined" value of the pipeACK variable.

WG draft 04 contained:

- o Editorial corrections.
- o Introduced the "cwnd-limited" term.
- o An adjustment to the procedure at the start of a cwnd-limited phase - the new text is intended to ensure that new-cwv is not unnecessarily more conservative than standard TCP when the flow is cwnd-limited. This resolves two issues: first it prevents pathologies in which pipeACK increases slowly and erratically. It also ensures that performance of bulk applications is not significantly impacted when using the method.
- o Clearly identifies that pacing (or equivalent) is required during the NVP to control burstiness. New section added.

WG draft 05 contained:

- o Clarification to first two bullets in section 4.4 describing cwnd-limited, to explain these are really alternates to the same case.
- o Section giving implementation examples was restructured to clarify there are two methods described.
- o Cross References to sections updated - thanks to comments from Martin Winbjoerk and Tim Wicinski.

10. References

10.1. 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.

- [RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window Validation", RFC 2861, June 2000.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 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.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.
- [RFC6928] Chu, J., Dukkupati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, April 2013.

10.2. Informative References

- [All05] and , "Notes on burst mitigation for transport protocols", March 2005.
- [Bis08] Biswas, and Fairhurst, "A Practical Evaluation of Congestion Window Validation Behaviour, 9th Annual Postgraduate Symposium in the Convergence of Telecommunications, Networking and Broadcasting (PGNet), Liverpool, UK", June 2008.
- [Bis10] Biswas, , Sathiaselalan, , Secchi, , and Fairhurst, "Analysing TCP for Bursty Traffic, Int'l J. of Communications, Network and System Sciences, 7(3)", June 2010.
- [Bis11] Biswas, , "PhD Thesis, Internet congestion control for variable rate TCP traffic, School of Engineering, University of Aberdeen", June 2011.
- [Fair12] Sathiaselalan, , Secchi, , Fairhurst, , and Biswas, "Enhancing TCP Performance to support Variable-Rate Traffic, 2nd Capacity Sharing Workshop, ACM CoNEXT, Nice, France, 10th December 2012.", June 2008.

- [Hug01] Hughes, , Touch, , and Heidemann, "Issues in TCP Slow-Start Restart After Idle (Work-in-Progress)", December 2001.
- [Liu07] Liu, , Allman, , Jiny, , and Wang, "Congestion Control without a Startup Phase, 5th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet), Los Angeles, California, USA", February 2007.

Authors' Addresses

Godred Fairhurst
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen, Scotland AB24 3UE
UK

Email: gorry@erg.abdn.ac.uk
URI: <http://www.erg.abdn.ac.uk>

Arjuna Sathiaselalan
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen, Scotland AB24 3UE
UK

Email: arjuna@erg.abdn.ac.uk
URI: <http://www.erg.abdn.ac.uk>

Raffaello Secchi
University of Aberdeen
School of Engineering
Fraser Noble Building
Aberdeen, Scotland AB24 3UE
UK

Email: raffaello@erg.abdn.ac.uk
URI: <http://www.erg.abdn.ac.uk>

TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Intended status: Experimental
Expires: March 21, 2014

P. Hurtig
A. Brunstrom
Karlstad University
A. Petlund
Simula Research Laboratory AS
M. Welzl
University of Oslo
September 17, 2013

TCP and SCTP RTO Restart
draft-ietf-tcpm-rtorestart-01

Abstract

This document describes a modified algorithm for managing the TCP and SCTP retransmission timers that provides faster loss recovery when there is a small amount of outstanding data for a connection. 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 March 21, 2014.

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.

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 [RFC5681]. 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 number 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 few 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.

While the reasons listed above are valid for any flow, the third reason is common for applications that transmit short flows, or use 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 risk of spurious timeout. Spurious timeouts typically degrade the performance of flows with multiple bursts of data, as a burst following a spurious timeout might not fit within the reduced congestion window (cwnd).

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 RTO_{min} 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. 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.

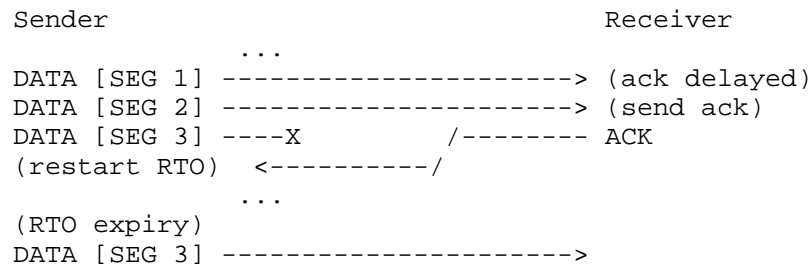


Figure 1: RTO restart example

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. 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."

Although fast retransmit is preferable there are 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 a sender-only modification to TCP and SCTP which updates 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.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

In this section, we discuss the applicability and a number of issues surrounding the modified RTO restart.

4.1. Applicability

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]. For applications that have strict timing requirements (e.g. interactive web and gaming) rather than throughput requirements, the modified restart approach could be important because the RTT and also the delayed ACK timer of receivers are often 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 by 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.

4.2. Spurious Timeouts

This document describes a modified RTO restart behavior that, in some situations, reduces the loss detection time and thereby increases the risk of spurious timeouts. In theory, the retransmission timer has a lower bound of 1 second [RFC6298], which limits the risk of having spurious timeouts. However, in practice most implementations use a significantly lower value. Initial measurements, conducted by the authors, show slight increases in the number of spurious timeouts when such lower values are used. However, further experiments, in different environments and with different types of traffic, are encouraged to quantify such increases more reliably.

Does a slightly increased risk matter? Generally, spurious timeouts have a negative effect on TCP/SCTP performance as the congestion window is reduced to one segment [RFC5681], limiting an application's ability to transmit large amounts of data instantaneously. However, with respect to RTO restart spurious timeouts are only a problem for applications transmitting multiple bursts of data within a single flow. Other types of flows, e.g. long-lived bulk flows, are not affected as the algorithm is only applied when the amount of outstanding segments is less than four and no previously unsent data is available. Furthermore, short-lived and application-limited flows are typically not affected as they are too short to experience the effect of congestion control or have a transmission rate that is quickly attainable.

While a slight increase in spurious timeouts has been observed using the modified RTO restart approach, it is not clear whether the effects of this increase mandate any future algorithmic changes or not -- especially since most modern operating systems already include mechanisms to detect [RFC3522][RFC3708][RFC5682] and resolve [RFC4015] possible problems with spurious retransmissions. Further experimentation is needed to determine this and thereby move this specification from experimental to proposed standard.

5. Related Work

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 has no buffer space. [RFC5827] specifies an early retransmit algorithm to enable fast loss recovery in such situations. By dynamically lowering the number 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.

Following the fast retransmit mechanism standardized in [RFC5681] this draft assumes a value of 3 for dupthresh. However, by considering a dynamic value for dupthresh a tighter integration with early retransmit (or other experimental algorithms) could also be possible.

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 normally two RTTs, and a spurious PTO is less risky than a spurious RTO because it would not have the same negative effects (clearing the scoreboard and restarting with slow-

start). In contrast, RTO restart is a small sender-only modification of the RTO management algorithm and does not require an additional timer or the use of SACK.

TLP is applicable 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 four and no new segments are available for transmission. The PTO has the same inherent restart problems as the RTO timer and could be combined with the modified restart approach to offer more consistent timeouts.

6. Acknowledgements

The authors wish to thank Godred Fairhurst, Yuchung Cheng, Mark Allman, Anantha Ramaiah and Richard Scheffenegger for commenting the draft and the ideas behind it.

All the authors are funded by the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700). The views expressed are solely those of the author(s).

7. IANA Considerations

This memo includes no request to IANA.

8. 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.

9. Changes from Previous Versions

9.1. Changes from draft-ietf-...-00 to -01

- o Improved the wording throughout the document.
- o Removed the possibility for a connection limited by the receiver's advertised window to use RTO restart, decreasing the risk of spurious retransmission timeouts.
- o Added a section that discusses the applicability of and problems related to the RTO restart mechanism.
- o Updated the text describing the relationship to TLP to reflect updates made in this draft.

- o Added acknowledgments.

10. References

10.1. 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.
- [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.
- [RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, February 2004.
- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.
- [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.
- [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.
- [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.

10.2. 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", Internet-draft draft-dukkipati-tcpm-tcp-
loss-probe-00.txt, 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

Anna Brunstrom
Karlstad University
Universitetsgatan 2
Karlstad 651 88
Sweden

Phone: +46 54 700 17 95
Email: anna.brunstrom@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 and Minor Extensions
(TCPM) WG
Internet-Draft
Obsoletes: 4614 (if approved)
Intended status: Informational
Expires: February 13, 2015

M. Duke
F5
R. Braden
ISI
W. Eddy
MTI Systems
E. Blanton

A. Zimmermann
NetApp, Inc.
August 12, 2014

A Roadmap for Transmission Control Protocol (TCP) Specification
Documents
draft-ietf-tcpm-tcp-rfc4614bis-08

Abstract

This document contains a "roadmap" to the Requests for Comments (RFC) documents relating to the Internet's Transmission Control Protocol (TCP). This roadmap provides a brief summary of the documents defining TCP and various TCP extensions that have accumulated in the RFC series. This serves as a guide and quick reference for both TCP implementers and other parties who desire information contained in the TCP-related RFCs.

This document obsoletes RFC 4614.

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 February 13, 2015.

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
2. Core Functionality	5
3. Strongly Encouraged Enhancements	8
3.1. Fundamental Changes	8
3.2. Congestion Control Extensions	9
3.3. Loss Recovery Extensions	11
3.4. Detection and Prevention of Spurious Retransmissions	12
3.5. Path MTU Discovery	13
3.6. Header Compression	14
3.7. Defending Spoofing and Flooding Attacks	15
4. Experimental Extensions	17
4.1. Architectural Guidelines	17
4.2. Fundamental Changes	18
4.3. Congestion Control Extensions	18
4.4. Loss Recovery Extensions	20
4.5. Detection and Prevention of Spurious Retransmissions	20
4.6. TCP Timeouts	21
4.7. Multipath TCP	21
5. TCP Parameters at IANA	22
6. Historic and Undeployed Extensions	23
7. Support Documents	26
7.1. Foundational Works	26
7.2. Architectural Guidelines	28
7.3. Difficult Network Environments	29
7.4. Guidance for Developing, Analyzing, and Evaluating TCP	32
7.5. Implementation Advice	33
7.6. Tools and Tutorials	35
7.7. MIB Modules	36
7.8. Case Studies	37
8. Undocumented TCP Features	38
9. Security Considerations	40
10. IANA Considerations	40
11. Acknowledgments	40
12. References	40
12.1. Normative References	40
12.2. Informative References	50
Authors' Addresses	51

1. Introduction

A correct and efficient implementation of the Transmission Control Protocol (TCP) is a critical part of the software of most Internet hosts. As TCP has evolved over the years, many distinct documents have become part of the accepted standard for TCP. At the same time, a large number of experimental modifications to TCP have also been published in the RFC series, along with informational notes, case studies, and other advice.

As an introduction to newcomers and an attempt to organize the plethora of information for old hands, this document contains a "roadmap" to the TCP-related RFCs. It provides a brief summary of the RFC documents that define TCP. This should provide guidance to implementers on the relevance and significance of the standards-track extensions, informational notes, and best current practices that relate to TCP.

This document is not an update of RFC 1122 [RFC1122] and is not a rigorous standard for what needs to be implemented in TCP. This document is merely an informational roadmap that captures, organizes, and summarizes most of the RFC documents that a TCP implementer, experimenter, or student should be aware of. Particular comments or broad categorizations that this document makes about individual mechanisms and behaviors are not to be taken as definitive, nor should the content of this document alone influence implementation decisions.

This roadmap includes a brief description of the contents of each TCP-related RFC. In some cases, we simply supply the abstract or a key summary sentence from the text as a terse description. In addition, a letter code after an RFC number indicates its category in the RFC series (see BCP 9 [RFC2026] for explanation of these categories):

S - Standards Track (Proposed Standard, Draft Standard, or Internet Standard)

E - Experimental

I - Informational

H - Historic

B - Best Current Practice

U - Unknown (not formally defined)

Note that the category of an RFC does not necessarily reflect its current relevance. For instance, RFC 5681 [RFC5681] is considered part of the required core functionality of TCP, although the RFC is only a Draft Standard. Similarly, some Informational RFCs contain significant technical proposals for changing TCP.

Finally, if an error in the technical content has been found after publication of an RFC, this fact is indicated by the term "(Errata)" in the headline of the RFC's description. The contents of the errata can be found at the RFC editor home page [Errata].

This roadmap is divided into three main sections. Section 2 lists the RFCs that describe absolutely required TCP behaviors for proper functioning and interoperability. Further RFCs that describe strongly encouraged, but non-essential, behaviors are listed in Section 3. Experimental extensions that are not yet standard practices, but that potentially could be in the future, are described in Section 4.

The reader will probably notice that these three sections are broadly equivalent to MUST/SHOULD/MAY specifications (per RFC 2119 [RFC2119]), and although the authors support this intuition, this document is merely descriptive; it does not represent a binding standards-track position. Individual implementers still need to examine the standards documents themselves to evaluate specific requirement levels.

Section 5 describes both the procedures that the Internet Assigned Numbers Authority (IANA) uses and an RFC author should follow when new TCP parameters are requested and finally assigned.

A small number of older experimental extensions that have not been widely implemented, deployed, and used are noted in Section 6. Many other supporting documents that are relevant to the development, implementation, and deployment of TCP are described in Section 7.

A small number of fairly ubiquitous important implementation practices that are not currently documented in the RFC series are listed in Section 8.

Within each section, RFCs are listed in the chronological order of their publication dates.

2. Core Functionality

A small number of documents compose the core specification of TCP. These define the required core functionalities of TCP's header

parsing, state machine, congestion control, and retransmission timeout computation. These base specifications must be correctly followed for interoperability.

RFC 793 S: "Transmission Control Protocol", STD 7 (September 1981) (Errata)

This is the fundamental TCP specification document [RFC0793]. Written by Jon Postel as part of the Internet protocol suite's core, it describes the TCP packet format, the TCP state machine and event processing, and TCP's semantics for data transmission, reliability, flow control, multiplexing, and acknowledgment.

Section 3.6 of RFC 793, describing TCP's handling of the IP precedence and security compartment, is mostly irrelevant today. RFC 2873 (see Section 2) changed the IP precedence handling, and the security compartment portion of the API is no longer implemented or used. In addition, RFC 793 did not describe any congestion control mechanism. Otherwise, however, the majority of this document still accurately describes modern TCPs. RFC 793 is the last of a series of developmental TCP specifications, starting in the Internet Experimental Notes (IENs) and continuing in the RFC series.

RFC 1122 S: "Requirements for Internet Hosts - Communication Layers" (October 1989)

This document [RFC1122] updates and clarifies RFC 793 (see Section 2), fixing some specification bugs and oversights. It also explains some features such as keep-alives and Karn's and Jacobson's RTO estimation algorithms [KP87][Jac88][JK92]. ICMP interactions are mentioned, and some tips are given for efficient implementation. RFC 1122 is an Applicability Statement, listing the various features that MUST, SHOULD, MAY, SHOULD NOT, and MUST NOT be present in standards-conforming TCP implementations. Unlike a purely informational "roadmap", this Applicability Statement is a standards document and gives formal rules for implementation.

RFC 2460 S: "Internet Protocol, Version 6 (IPv6) Specification" (December 1998) (Errata)

This document [RFC2460] is of relevance to TCP because it defines how the pseudo-header for TCP's checksum computation is derived when 128-bit IPv6 addresses are used instead of 32-bit IPv4 addresses. Additionally, RFC 2675 (see Section 3.1) describes TCP changes required to support IPv6 jumbograms.

RFC 2873 S: "TCP Processing of the IPv4 Precedence Field" (June 2000) (Errata)

This document [RFC2873] removes from the TCP specification all processing of the precedence bits of the TOS byte of the IP header. This resolves a conflict over the use of these bits between RFC 793 Section 2 and Differentiated Services [RFC2474].

RFC 5681 S: "TCP Congestion Control" (August 2009)

Although RFC 793 (see Section 2) did not contain any congestion control mechanisms, today congestion control is a required component of TCP implementations. This document [RFC5681] defines congestion avoidance and control mechanism for TCP, based on Van Jacobson's 1988 SIGCOMM paper [Jac88].

A number of behaviors that together constitute what the community refers to as "Reno TCP" is described in RFC 5681. The name "Reno" comes from the Net/2 release of the 4.3 BSD operating system. This is generally regarded as the least common denominator among TCP flavors currently found running on Internet hosts. Reno TCP includes the congestion control features of slow start, congestion avoidance, fast retransmit, and fast recovery.

RFC 5681 details the currently accepted congestion control mechanism, while RFC 1122 Section 2 mandates that such a congestion control mechanism must be implemented. RFC 5681 differs slightly from the other documents listed in this section, as it does not affect the ability of two TCP endpoints to communicate; however, congestion control remains a critical component of any widely deployed TCP implementation and is required for the avoidance of congestion collapse and to ensure fairness among competing flows.

RFC 2001 and RFC 2581 are the conceptual precursors of RFC 5681. The most important changes relative to RFC 2581 are:

- (a) The initial window requirements were changed to allow larger Initial Windows as standardized in [RFC3390] (see Section 3.2).
- (b) During slow start and congestion avoidance, the usage of Appropriate Byte Counting [RFC3465] (see Section 3.2) is explicitly recommended.
- (c) The use of Limited Transmit [RFC3042] (see Section 3.3) is now recommended.

RFC 6093 S: "On the Implementation of the TCP Urgent Mechanism" (January 2011)

This document [RFC6093] analyzes how current TCP stacks process TCP urgent indications, and how the behavior of widely deployed middleboxes affects the urgent indications processing. The document updates the relevant specifications such that it accommodates current practice in processing TCP urgent indications. Finally, the document raises awareness about the reliability of TCP urgent indications in the Internet, and recommends against the use of urgent mechanism.

RFC 6298 S: "Computing TCP's Retransmission Timer" (June 2011)

Abstract: "This document defines the standard algorithm that Transmission Control Protocol (TCP) senders are required to use to compute and manage their retransmission timer. It expands on the discussion in section 4.2.3.1 of RFC 1122 (see Section 2) and upgrades the requirement of supporting the algorithm from a SHOULD to a MUST." [RFC6298]. RFC 6298 updates RFC 2988 by changing the initial RTO from 3s to 1s

RFC 6691 I: "TCP Options and Maximum Segment Size (MSS)" (July 2012)

This document [RFC6691] clarifies what value to use with the TCP Maximum Segment Size (MSS) option when IP and TCP options are in use.

3. Strongly Encouraged Enhancements

This section describes recommended TCP modifications that improve performance and security. Section 3.1 represents fundamental changes to the protocol. Section 3.2 and Section 3.3 list improvements over the congestion control and loss recovery mechanisms as specified in RFC 5681 (see Section 2). Section 3.4 describes algorithms that allow a TCP sender to detect whether it has entered loss recovery spuriously. Section 3.5 comprises Path MTU Discovery mechanisms. Schemes for TCP/IP header compression are listed in Section 3.6. Finally, Section 3.7 deals with the problem of preventing acceptance of forged segments and flooding attacks.

3.1. Fundamental Changes

RFCs 2675 and 7323 represent fundamental changes to TCP by redefining how parts of the basic TCP header and options are interpreted. RFC 7323 defines the Window Scale Option, which re-interprets the advertised receive window. RFC 2675 specifies that MSS option and

urgent pointer fields with a value of 65,535 are to be treated specially.

RFC 2675 S: "IPv6 Jumbograms" (August 1999) (Errata)

IPv6 supports longer datagrams than were allowed in IPv4. These are known as jumbograms, and use with TCP has necessitated changes to the handling of TCP's MSS and Urgent fields (both 16 bits). This document [RFC2675] explains those changes. Although it describes changes to basic header semantics, these changes should only affect the use of very large segments, such as IPv6 jumbograms, which are currently rarely used in the general Internet.

Supporting the behavior described in this document does not affect interoperability with other TCP implementations when IPv4 or non-jumbogram IPv6 is used. This document states that jumbograms are to only be used when it can be guaranteed that all receiving nodes, including each router in the end-to-end path, will support jumbograms. If even a single node that does not support jumbograms is attached to a local network, then no host on that network may use jumbograms. This explains why jumbogram use has been rare, and why this document is considered a performance optimization and not part of TCP over IPv6's basic functionality.

RFC 7323 S: "TCP Extensions for High Performance" (July 2014)

This document [I-D.ietf-tcpm-1323bis] defines TCP extensions for window scaling, timestamps, and protection against wrapped sequence numbers, for efficient and safe operation over paths with large bandwidth-delay products. These extensions are commonly found in currently used systems. The predecessor of this document, RFC 1323, was published in 1992, and is deployed in most TCP implementations. This document includes fixes and clarifications based on the gained deployment experience. One specific issued addressed in this specification is a recommendation how to modify the algorithm for estimating the mean RTT when timestamps are used. RFC 1072, RFC 1185, and RFC 1323 are the conceptual precursors of RFC 7323.

3.2. Congestion Control Extensions

Two of the most important aspects of TCP are its congestion control and loss recovery features. TCP treats lost packets as indicating congestion-related loss, and cannot distinguish between congestion-related loss and loss due to transmission errors. Even when ECN is in use, there is a rather intimate coupling between congestion control and loss recovery mechanisms. There are several extensions

to both features, and more often than not, a particular extension applies to both. In these two sub-sections, we group enhancements to TCP's congestion control, while the next sub-section focus on TCP's loss recovery.

RFC 3168 S: "The Addition of Explicit Congestion Notification (ECN) to IP" (September 2001)

This document [RFC3168] defines a means for end hosts to detect congestion before congested routers are forced to discard packets. Although congestion notification takes place at the IP level, ECN requires support at the transport level (e.g., in TCP) to echo the bits and adapt the sending rate. This document updates RFC 793 (see Section 2) to define two previously unused flag bits in the TCP header for ECN support. RFC 3540 (see Section 4.3) provides a supplementary (experimental) means for more secure use of ECN, and RFC 2884 (see Section 7.8) provides some sample results from using ECN.

RFC 3390 S: "Increasing TCP's Initial Window" (October 2002)

This document [RFC3390] specifies an increase in the permitted initial window for TCP from one segment to three or four segments during the slow start phase, depending on the segment size.

RFC 3465 E: "TCP Congestion Control with Appropriate Byte Counting (ABC)" (February 2003)

This document [RFC3465] suggests that congestion control use the number of bytes acknowledged instead of the number of acknowledgments received. This change improves the performance of TCP in situations where is no one-to-one relationship between data segments and acknowledgments (e.g. delayed ACKs or ACK loss) and closes a security hole TCP receivers can use to induce the sender into increasing the sending rate too rapidly (ACK-division [SCWA99][RFC3449]). ABC is recommended by RFC 5681 (see Section 2).

RFC 6633 S: "Deprecation of ICMP Source Quench Messages" (May 2012)

This document [RFC6633] formally deprecates the use of ICMP Source Quench messages by transport protocols and recommends against the implementation of [RFC1016].

3.3. Loss Recovery Extensions

For the typical implementation of the TCP fast recovery algorithm described in RFC 5681 (see Section 2), a TCP sender only retransmits a segment after a retransmit timeout has occurred, or after three duplicate ACKs have arrived triggering the fast retransmit. A single RTO might result in the retransmission of several segments, while the fast retransmit algorithm in RFC 5681 leads only to a single retransmission. Hence, multiple losses from a single window of data can lead to a performance degradation. Documents listed in this section aim to improve the overall performance of TCP's standard loss recovery algorithms. In particular, some of them allow TCP senders to recover more effectively when multiple segments are lost from a single flight of data.

RFC 2018 S: "TCP Selective Acknowledgment Options" (October 1996)
(Errata)

When more than one packet is lost during one round trip time TCP may experience poor performance since a TCP sender can only learn about a single lost packet per round trip time from cumulative acknowledgments. This document [RFC2018] defines the basic selective acknowledgment (SACK) mechanism for TCP, which can help to overcome these limitations. The receiving TCP returns SACK blocks to inform the sender which data has been received. The sender can then retransmit only the missing data segments.

RFC 3042 S: "Enhancing TCP's Loss Recovery Using Limited Transmit"
(January 2001)

Abstract: "This document proposes Limited Transmit, a new Transmission Control Protocol (TCP) mechanism that can be used to more effectively recover lost segments when a connection's congestion window is small, or when a large number of segments are lost in a single transmission window." [RFC3042] Tests from 2004 showed that Limited Transmit was deployed in roughly one third of the web servers tested [MAF04]. Limited Transmit is recommended by RFC 5681 (see Section 2).

RFC 6582 S: "The NewReno Modification to TCP's Fast Recovery Algorithm" (April 2012)

This document [RFC6582] specifies a modification to the standard Reno fast recovery algorithm, whereby a TCP sender can use partial acknowledgments to make inferences determining the next segment to send in situations where SACK would be helpful but isn't available. Although it is only a slight modification, the NewReno behavior can make a significant difference in performance when

multiple segments are lost from a single window of data.

RFC 2582 and RFC 3782 are the conceptual precursors of RFC 6582. The main change in RFC 3782 relative to RFC 2582 was to specify the Careful variant of NewReno's Fast Retransmit and Fast Recovery algorithms and advance those two algorithms from Experimental to Standards Track status. The main change in RFC 6582 relative to RFC 3782 was to solve a performance degradation that could occur if FlightSize on Full ACK reception is zero.

RFC 6675 S: "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP" (August 2012)

This document [RFC6675] describes a conservative loss recovery algorithm for TCP that is based on the use of the selective acknowledgment (SACK) TCP option [RFC2018] (see Section 3.3). The algorithm conforms to the spirit of the congestion control specification in RFC 5681 (see Section 2), but allows TCP senders to recover more effectively when multiple segments are lost from a single flight of data.

RFC 6675 is a revision of RFC 3517 to address several situations that are not handled explicitly before. In particular

- (a) it improves the loss detection in the event that the sender has outstanding segments that are smaller than SMSS.
- (b) it modifies the definition of a "duplicate acknowledgment" to utilize the SACK information in detecting loss.
- (c) it maintains the ACK clock under certain circumstances involving loss at the end of the window.

3.4. Detection and Prevention of Spurious Retransmissions

Spurious retransmission timeouts are harmful to TCP performance and multiple algorithms have been defined for detecting when spurious retransmissions have occurred, and then responding differently in order to recover performance. The IETF defined multiple algorithms because there are tradeoffs in whether or not certain TCP options need to be implemented, and concerns about IPR status. The Standards Track documents in this section are closely related to the Experimental documents in Section 4.5 also addressing this topic.

RFC 2883 S: "An Extension to the Selective Acknowledgement (SACK) Option for TCP" (July 2000)

This document [RFC2883] extends RFC 2018 (see Section 3.3). It enables use of the SACK option to acknowledge duplicate packets. With this extension, called DSACK, the sender is able to infer the order of packets received at the receiver, and therefore to infer

when it has unnecessarily retransmitted a packet. A TCP sender could then use this information to detect spurious retransmissions (see [RFC3708]).

RFC 4015 S: "The Eifel Response Algorithm for TCP" (February 2005)

This document [RFC4015] describes the response portion of the Eifel algorithm, which can be used in conjunction with one of several methods of detecting when loss recovery has been spuriously entered, such as the Eifel detection algorithm in RFC 3522 (see Section 4.5), the algorithm in RFC 3708 (see Section 4.5), or F-RTO in RFC 5682 (see Section 3.4).

Abstract: "Based on an appropriate detection algorithm, the Eifel response algorithm provides a way for a TCP sender to respond to a detected spurious timeout. It adapts the retransmission timer to avoid further spurious timeouts, and can avoid - depending on the detection algorithm - the often unnecessary go-back-N retransmits that would otherwise be sent. In addition, the Eifel response algorithm restores the congestion control state in such a way that packet bursts are avoided."

RFC 5682 S: "Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP" (September 2009)

The F-RTO detection algorithm [RFC5682], originally described in RFC 4138, provides an option for inferring spurious retransmission timeouts. Unlike some similar detection methods (e.g. RFC 3522 in Section 4.5 and RFC 3708 in Section 4.5), F-RTO does not rely on the use of any TCP options. The basic idea is to send previously unsent data after the first retransmission after a RTO. If the ACKs advance the window, the RTO may be declared spurious.

3.5. Path MTU Discovery

The MTUs supported by different links and tunnels within the Internet can vary widely. Fragmentation of packets larger than the supported MTU on a hop is undesirable. As TCP is the segmentation layer for dividing an application's bytestream into IP packet payloads, TCP implementations generally include Path MTU Discovery (PMTUD) mechanisms in order to maximize the size of segments they send, without causing fragmentation within the network. Some algorithms may utilize signaling from routers on the path that the MTU has been exceeded.

RFC 1191 S: "Path MTU Discovery" (November 1990)

Abstract: "This memo describes a technique for dynamically discovering the MTU of an arbitrary Internet path. It specifies a small change to the way routers generate one type of ICMP message. For a path that passes through a router that has not been so changed, this technique might not discover the correct path MTU, but it will always choose a path MTU as accurate as, and in many cases more accurate than, the path MTU that would be chosen by current practice." [RFC1191]

RFC 1981 S: "Path MTU Discovery for IP version 6" (August 1996)

Abstract: "This document describes Path MTU Discovery for IP version 6. It is largely derived from RFC 1191 (see Section 3.5), which describes Path MTU Discovery for IP version 4." [RFC1981]

RFC 4821 S: "Packetization Layer Path MTU Discovery" (March 2007)

Abstract: "This document describes a robust method for Path MTU Discovery (PMTUD) that relies on TCP or some other Packetization Layer to probe an Internet path with progressively larger packets. This method is described as an extension to RFC 1191 (see Section 3.5) and RFC 1981 (see Section 3.5), which specify ICMP-based Path MTU Discovery for IP versions 4 and 6, respectively." [RFC4821]

3.6. Header Compression

Especially in streaming applications, the overhead of TCP/IP headers could correspond to more than 50% of the total amount of data sent. Such large overheads may be tolerable in wired LANs where capacity is often not an issue, but are excessive for WANs and wireless systems where bandwidth is scarce. Header compression schemes for TCP/IP like "RObust Header Compression (ROHC) can significantly compress this overhead. It performs well over links with significant error rates and long round-trip times.

RFC 1144 S: "Compressing TCP/IP Headers for Low-Speed Serial Links" (February 1990)

This document [RFC1144] describes a method for compressing the headers of TCP/IP datagrams to improve performance over low speed serial links. The method described in this document is limited in its handling of TCP options and cannot compress the headers of SYNs and FINs.

RFC 6846 S: "RObust Header Compression (ROHC): A Profile for TCP/IP (ROHC-TCP)" January 2013)

From abstract: "This document specifies a RObust Header Compression (ROHC) profile for compression of TCP/IP packets. The profile, called ROHC-TCP, provides efficient and robust compression of TCP headers, including frequently used TCP options such as selective acknowledgments (SACKs) and Timestamps." [RFC6846] RFC 6846 is the successor of RFC 4996. It fixes a technical issue with the SACK compression and clarifies other compression methods used.

3.7. Defending Spoofing and Flooding Attacks

By default, TCP lacks any cryptographic structures to differentiate legitimate segments from those spoofed from malicious hosts. Spoofing valid segments requires correctly guessing a number of fields. The documents in this sub-section describe ways to make that guessing harder, or to prevent it from being able to affect a connection negatively.

RFC 4953 I: "Defending TCP Against Spoofing Attacks" (July 2007)

This document [RFC4953] discusses the recently increased vulnerability of long-lived TCP connections, such as BGP connections, to reset (send RST) spoofing attacks. The document analyzes the vulnerability, discussing proposed solutions at the transport level and their inherent challenges, as well as existing network level solutions and the feasibility of their deployment.

RFC 5461 I: "TCP's Reaction to Soft Errors" (February 2009)

This document [RFC5461] describes a non-standard but widely implemented modification to TCP's handling of ICMP soft error messages that rejects pending connection-requests when such error messages are received. This behavior reduces the likelihood of long delays between connection-establishment attempts that may arise in some scenarios.

RFC 4987 I: "TCP SYN Flooding Attacks and Common Mitigations" (August 2007)

This document [RFC4987] describes the well-known TCP SYN flooding attack. It analyzes and discusses various countermeasures against these attacks, including their use and trade-offs.

RFC 5925 S: "The TCP Authentication Option" (May 2010)

This document [RFC5925] describes the TCP Authentication Option (TCP-AO), which is used to authenticate TCP segments. TCP-AO obsoletes the TCP MD5 Signature option of RFC 2385. It supports the use of stronger hash functions, protects against replays for long-lived TCP connections (as used, e.g., in BGP and LDP), coordinates key exchanges between endpoints, and provides a more explicit recommendation for external key management. Cryptographic algorithms for TCP-AO are defined in [RFC5926] (see Section 3.7).

RFC 5926 S: "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)" (May 2010)

This document [RFC5926] specifies the algorithms and attributes that can be used in TCP Authentication Option's (TCP-AO) [RFC5925] (see Section 3.7) current manual keying mechanism and provides the interface for future message authentication codes (MACs).

RFC 5927 I: "ICMP attacks against TCP" (July 2010)

Abstract: "This document discusses the use of the Internet Control Message Protocol (ICMP) to perform a variety of attacks against the Transmission Control Protocol (TCP). Additionally, this document describes a number of widely implemented modifications to TCP's handling of ICMP error messages that help to mitigate these issues." [RFC5927]

RFC 5961 S: "Improving TCP's Robustness to Blind In-Window Attacks" (August 2010)

This document [RFC5961] describes minor modifications to how TCP handles inbound segments. This renders TCP connections, especially long-lived connections such as H-323 or BGP, less vulnerable to spoofed packet injection attacks where the 4-tuple (the source and destination IP addresses and the source and destination ports) has been guessed.

RFC 6528 S: "Defending Against Sequence Number Attacks" (February 2012)

Abstract: "This document [RFC6528] specifies an algorithm for the generation of TCP Initial Sequence Numbers (ISNs), such that the chances of an off-path attacker guessing the sequence numbers in use by a target connection are reduced. This document revises (and formally obsoletes) RFC 1948, and takes the ISN generation algorithm originally proposed in that document to Standards Track,

formally updating RFC 793 (see Section 2).

4. Experimental Extensions

The RFCs in this section are either experimental and may become proposed standards in the future or are proposed standard (or informational), but can be considered as experimental due to lack of wide deployment. At least part of the reason that they are still experimental is to gain more wide-scale experience with them before a standards track decision is made.

If the experimental RFC is a proposal for a new protocol capability or service, i.e., it requires a new TCP option code point, the implementation and experimentation should follow [RFC6994] (see Section 5), which describes how the experimental TCP option code points can concurrently support multiple TCP extensions.

By their publication as experimental RFCs, it is hoped that the community of TCP researchers will analyze and test the contents of these RFCs. Although experimentation is encouraged, there is not yet formal consensus that these are fully logical and safe behaviors. Wide-scale deployment of implementations that use these features should be well thought-out in terms of consequences.

4.1. Architectural Guidelines

As multiple flows may share the same paths, sections of paths, or other resources, the TCP implementation may benefit from sharing information across TCP connections or other flows. Some Experimental proposals have been documented and some implementations have included the concepts.

RFC 2140 I: "TCP Control Block Interdependence" (April 1997)

This document [RFC2140] suggests how TCP connections between the same endpoints might share information, such as their congestion control state. To some degree, this is done in practice by a few operating systems; for example, Linux currently has a destination cache. Although this RFC is technically informational, the concepts it describes are in experimental use, so we include it in this section.

RFC 3124 S: "The Congestion Manager" (June 2001)

This document [RFC3124], the Congestion Manager, is a related proposal to RFC 2140 (see Section 4.1). The idea behind the Congestion Manager, moving congestion control outside of

individual TCP connections, represents a modification to the core of TCP, which supports sharing information among TCP connections. Although a Proposed Standard, some pieces of the Congestion Manager support architecture have not been specified yet, and it has not achieved use or implementation beyond experimental stacks, so it is not listed among the standard TCP enhancements in this roadmap.

4.2. Fundamental Changes

Like the standard documents listed in Section 3.1, there also exist new Experimental RFCs that specify fundamental changes to TCP. At the time of writing, the only example so far is TCP Fast Open that deviates from the standard TCP semantics of [RFC0793].

RFC XXX E: "TCP Fast Open" (XXX 2014)

This document [I-D.ietf-tcpm-fastopen] describes TCP Fast Open that allows data to be carried in the SYN and SYN-ACK packets and consumed by the receiver during the initial connection handshake. It saves up to one RTT compared to the standard TCP, which requires a three-way handshake to complete before data can be exchanged.

4.3. Congestion Control Extensions

TCP congestion control has been an extremely active research area for many years (see RFC 5783, Section 7.6), as it determines the performance of many applications that use TCP. A number of experimental RFCs address issues with flow start-up, overshoot, and steady-state behavior in the basic RFC 5681 (see Section 2) algorithms. In these sub-sections, enhancements to TCP's congestion control are listed. The next sub-section focuses on TCP's loss recovery.

RFC 2861 E: "TCP Congestion Window Validation" (June 2000)

This document [RFC2861] suggests reducing the congestion window over time when no packets are flowing. This behavior is more aggressive than that specified in RFC 5681 (see Section 2), which says that a TCP sender SHOULD set its congestion window to the initial window after an idle period of an RTO or greater.

RFC 3540 E: "Robust Explicit Congestion Notification (ECN) signaling with Nonces" (June 2003)

This document [RFC3540] describes an optional addition to ECN that protects against accidental or malicious concealment of marked

packets from the TCP sender.

RFC 3649 E: "HighSpeed TCP for Large Congestion Windows" (December 2003)

This document [RFC3649] proposes a modification to TCP's congestion control mechanism for use with TCP connections with large congestion windows, to allow TCP to achieve a higher throughput in high-bandwidth environments.

RFC 3742 E: "Limited Slow-Start for TCP with Large Congestion Windows" (March 2004)

This document [RFC3742] describes a more conservative slow-start behavior to prevent massive packet losses when a connection uses a very large congestion window.

RFC 4782 E: "Quick-Start for TCP and IP" (January 2007) (Errata)

This document [RFC4782] specifies the optional Quick-Start mechanism for TCP. This mechanism allows connections to use higher sending rates at the beginning of the data transfer or after an idle period, provided that there is significant unused bandwidth along the path, and the sender and all of the routers along the path approve this higher rate.

RFC 5562 E: "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets" (June 2009)

This document [RFC5562] describes an experimental modification to ECN [RFC3168] (see Section 3.2) for the use of ECN in TCP SYN/ACK packets. This would allow to ECN-mark rather than drop the TCP SYN/ACK packet at an ECN-capable router, and to avoid the severe penalty of a retransmission timeout for a connection when the SYN/ACK packet is dropped.

RFC 5690 I: "Adding Acknowledgement Congestion Control to TCP" (February 2010)

This document [RFC5690] describes a congestion control mechanism for acknowledgment (ACKs) traffic in TCP. The mechanism is based on the acknowledgment congestion control of the Datagram Congestion Control Protocol's (DCCP's) [RFC4340] Congestion Control Identifier (CCID) 2 [RFC4341].

RFC 6928 E: "Increasing TCP's Initial Window" (April 2013)

This document [RFC6928] proposes to increase the TCP initial window from between 2 and 4 segments, as specified in RFC 3390 (see Section 3.2), to 10 segments with a fallback to the existing recommendation when performance issues are detected.

4.4. Loss Recovery Extensions

RFC 5827 E: "Early Retransmit for TCP and SCTP" (April 2010)

This document [RFC5827] proposes the "Early Retransmit" mechanism for TCP (and SCTP) that can be used to recover lost segments when a connection's congestion window is small. In certain special circumstances, Early Retransmit reduces the number of duplicate acknowledgments required to trigger fast retransmit to recover segment losses without waiting for a lengthy retransmission timeout.

RFC 6069 E: "Making TCP more Robust to Long Connectivity Disruptions (TCP-LCD)" (December 2010)

This document [RFC6069] describes how standard ICMP messages can be used to disambiguate true congestion loss from non-congestion loss caused by connectivity disruptions. It proposes a reversion strategy of TCP's retransmission timer that enables a more prompt detection of whether or not the connectivity has been restored.

RFC 6937 E: "Proportional Rate Reduction for TCP" (May 2013)

This document [RFC6937] describes an experimental Proportional Rate Reduction (PRR) algorithm as an alternative to the widely deployed Fast Recovery algorithm, to improve the accuracy of the amount of data sent by TCP during loss recovery.

4.5. Detection and Prevention of Spurious Retransmissions

In addition to the Standards Track extensions to deal with spurious retransmissions in Section 3.4, Experimental proposals have also been documented.

RFC 3522 E: "The Eifel Detection Algorithm for TCP" (April 2003)

The Eifel detection algorithm [RFC3522] allows a TCP sender to detect a posteriori whether it has entered loss recovery unnecessarily by using the TCP timestamp option to solve the ACK ambiguity.

RFC 3708 E: "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions" (February 2004)

Abstract: "TCP and Stream Control Transmission Protocol (SCTP) provide notification of duplicate segment receipt through Duplicate Selective Acknowledgement (DSACKs) and Duplicate Transmission Sequence Number (TSN) notification, respectively. This document presents conservative methods of using this information to identify unnecessary retransmissions for various applications." [RFC3708]

RFC 4653 E: "Improving the Robustness of TCP to Non-Congestion Events" (August 2008)

In the presence of non-congestion events, such as reordering an out-of-order segment does not necessarily indicates a lost segment and congestion. This document [RFC4653] proposes to increase the threshold used to trigger a fast retransmission from the fixed value of three duplicate ACKs to about one congestion window of data in order to disambiguate true segment loss from segment reordering.

4.6. TCP Timeouts

Besides the well-known retransmission timeout the TCP standard [RFC0793] defines other timeouts. This section lists documents that deal with TCP's various timeouts.

RFC 5482 S: "TCP User Timeout Option" (June 2009)

As a local per-connection parameter the TCP user timeout controls how long transmitted data may remain unacknowledged before a connection is forcefully closed. This document [RFC5482] specifies the TCP User Timeout Option that allows one end of a TCP connection to advertise its current user timeout value. This information provides advice to the other end of the TCP connection to adapt its user timeout accordingly.

4.7. Multipath TCP

MultiPath TCP (MPTCP) is an ongoing effort within the IETF that allows a TCP connection to simultaneously use multiple IP-addresses/interfaces to spread their data across several subflows, while presenting a regular TCP interface to applications. Benefits of this include better resource utilization, better throughput and smoother reaction to failures. The documents listed in this section specify

the Multipath TCP scheme, while the documents in Sections 7.2, 7.4, and 7.5 provide some additional background information.

RFC 6356 E: "Coupled Congestion Control for Multipath Transport Protocols" (August 2011)

This document [RFC6356] presents a congestion control algorithm for multipath transport protocols such as Multipath TCP. It couples the congestion control algorithms running on different subflows by linking their increase functions, and dynamically controls the overall aggressiveness of the multipath flow. The result is an algorithm that is fair to TCP at bottlenecks while moving traffic away from congested links.

RFC 6824 E: "TCP Extensions for Multipath Operation with Multiple Addresses" (January 2013) (Errata)

This document [RFC6824] presents protocol changes required to add multipath capability to TCP; specifically, those for signaling and setting up multiple paths ("subflows"), managing these subflows, reassembly of data, and termination of sessions.

5. TCP Parameters at IANA

RFCs listed here describes both the procedures that the Internet Assigned Numbers Authority (IANA) uses when handling assignments and the procedures an RFC author should follow when requesting new TCP option codepoints.

RFC 2780 B: "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers" (March 2000)

Abstract: "This memo provides guidance for the IANA to use in assigning parameters for fields in the IPv4, IPv6, ICMP, UDP and TCP protocol headers." [RFC2780]

RFC 4727 S: "Experimental Values" (November 2006)

This document [RFC4727] reserves both TCP options 253 and 254 for experimentation purposes. When such experiments are deployed in the Internet, they should follow the additional requirements in RFC 6994 (see Section 5).

RFC 6335 B: "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry" (August 2011)

From abstract: "This document defines the procedures that the Internet Assigned Numbers Authority (IANA) uses when handling assignment and other requests related to the Service Name and Transport Protocol Port Number registry." [RFC6335]

RFC 6994 S: "Shared Use of Experimental TCP Options (August 2013)

This document [RFC6994] describes how the experimental TCP option code points can concurrently support multiple TCP extensions, even within the same connection. It creates an IANA registry for extensions to the experimental code points.

6. Historic and Undeployed Extensions

The RFCs listed here define extensions that have thus far failed to arouse substantial interest from implementers and have never seen widespread deployment, or were found to be defective for general use. Most of them are reclassified by [RFC6247] to Historic status.

RFC 721 U: "Out-of-Band Control Signals in a Host-to-Host Protocol" (September 1976): lack of interest

RFC 721 [RFC0721] addresses the problem of implementing a reliable out-of-band signal (interrupts) for use in a host-to-host protocol. The proposal was not included in the final TCP specification.

RFC 1078 U: "TCP Port Service Multiplexer (TCPMUX)" (November 1988): lack of interest

This document [RFC1078] proposes a protocol to contact multiple services on a single well-known TCP port using a service name instead of a well-known number.

RFC 1106 H: "TCP Big Window and NAK Options" (June 1989): found defective

This RFC [RFC1106] defined an alternative to the Window Scale option for using large windows and described the "negative acknowledgment" or NAK option. There is a comparison of NAK and SACK methods, and early discussion of TCP over satellite issues. RFC 1110 (see Section 6) explains some problems with the approaches described in RFC 1106. The options described in this document have not been adopted by the larger community, although NAKs are used in the SCPS-TP adaptation of TCP for satellite and spacecraft use, developed by the Consultative Committee for Space Data Systems (CCSDS).

RFC 1110 H: "A Problem with the TCP Big Window Option" (August 1989): deprecates RFC 1106

Abstract: "The TCP Big Window option discussed in RFC 1106 (see Section 6) will not work properly in an Internet environment which has both a high bandwidth * delay product and the possibility of disordering and duplicating packets. In such networks, the window size must not be increased without a similar increase in the sequence number space. Therefore, a different approach to big windows should be taken in the Internet." [RFC1110]

RFC 1146 H: "TCP Alternate Checksum Options" (March 1990): lack of interest

This document [RFC1146] defined more robust TCP checksums than the 16-bit ones-complement in use today. A typographical error in RFC 1145 is fixed in RFC 1146; otherwise, the documents are the same.

RFC 1263 I: "TCP Extensions Considered Harmful" (October 1991): lack of interest

This document [RFC1263] argues against "backwards compatible" TCP extensions. Specifically mentioned are several TCP enhancements that have been successful, including timestamps, window scaling, PAWS, and SACK. RFC 1263 presents an alternative approach called "protocol evolution", whereby several evolutionary versions of TCP would exist on hosts. These distinct TCP versions would represent upgrades to each other and could be header-incompatible. Interoperability would be provided by having a virtualization layer select the right TCP version for a particular connection. This idea did not catch on with the community, while the type of extensions RFC 1263 specifically targeted as harmful did become popular.

RFC 1379 H: "Extending TCP for Transactions -- Concepts" (November 1992): found defective

See RFC 1644, Section 6.

RFC 1644 H: "T/TCP -- TCP Extensions for Transactions Functional Specification" (July 1994): found defective

The inventors of TCP believed that cached connection state could have been used to eliminate TCP's 3-way handshake, to support two-packet request/response exchanges. RFC 1379 [RFC1379] (see Section 6) and RFC 1644 [RFC1644] show that this is far from simple. Furthermore, T/TCP floundered on the ease of denial-of-service attacks that can result. One idea pioneered by T/TCP

lives on in RFC 2140 (see Section 4.1), in the sharing of state across connections.

RFC 1693 H: "An Extension to TCP: Partial Order Service" (November 1994): lack of interest

This document [RFC1693] defines a TCP extension for applications that do not care about the order in which application-layer objects are received. Examples are multimedia and database applications. In practice, these applications either accept the possible performance loss because of TCP's strict ordering or they use specialized transport protocols other than TCP, such as PR-SCTP [RFC3758].

RFC 1705 I: "Six Virtual Inches to the Left: The Problem with IPng" (October 1994): lack of interest

To overcome the exhaustion of the IP class B address space, suggest this document [RFC1705] that a new version of TCP (TCPng) needs to be developed and deployed. It proposes that a globally unique address be assigned to Transport layer to uniquely identify an Internet host without specifying any routing information. Later work on splitting locator and identifier values is summarized well in [RFC6115], but no resulting changes to TCP have occurred.

RFC 6013 E: "TCP Cookie Transactions (TCPCT)" (January 2011): lack of interest

This document [RFC6013] describes a method to exchange a cookie (nonce) during the connection establishment to negotiate elimination of receiver state. These cookies are later used to inhibit premature closing of connections, and reduce retention of state after the connection has terminated.

Since the cookie pair is too large to fit with the other TCP options in the 40 bytes of TCP option space, the document further describes a method to extend the option space after the connection establishment.

Although RFC 6013 was published in 2011, the authors of this document places it in this section of the roadmap document due to two factors.

- (a) The authors are not aware of any wide deployment and use of RFC 6013.

- (b) RFC 6013 uses experimental TCP option codepoints, which prohibits a large-scale deployment.

7. Support Documents

This section contains several classes of documents that do not necessarily define current protocol behaviors, but that are nevertheless of interest to TCP implementers. Section 7.1 describes several foundational RFCs that give modern readers a better understanding of the principles underlying TCP's behaviors and development over the years. Section 7.2 contains architectural guidelines and principles for TCP architects and designers. The documents listed in Section 7.3 provide advice on using TCP in various types of network situations that pose challenges above those of typical wired links. Guidance for developing, analyzing, and evaluating TCP is given in Section 7.4. Some implementation notes and implementation advice can be found in Section 7.5. RFCs that describe tools for testing and debugging TCP implementations or that contain high-level tutorials on the protocol are listed in Section 7.6. The TCP Management Information Bases are described in Section 7.7, and Section 7.8 lists a number of case studies that have explored TCP performance.

7.1. Foundational Works

The documents listed in this section contain information that is largely duplicated by the standards documents previously discussed. However, some of them contain a greater depth of problem statement explanation or other context. Particularly, RFCs 813 - 817 (known as the "Dave Clark Five") describe some early problems and solutions (RFC 815 only describes the reassembly of IP fragments and is not included in this TCP roadmap).

RFC 675 U: "Specification of Internet Transmission Control Program" (December 1974)

This document [RFC0675] is a very early precursor of the fundamental RFC 793 (see Section 2), which already contained the three-way handshake in its final form and the concept of sliding windows for reliable data transmission. Apart from that the segment layout is totally different and the specified API differs from the latter RFC 793 (see Section 2).

RFC 761 U: "DoD standard Transmission Control Protocol" (January 1980)

This document [RFC0761] is the immediate precursor of RFC 793 (see

Section 2). The header format, the connection establishment including the different connection states, and the overall API correspond mostly to the final Standard RFC 793 (see Section 2).

RFC 813 U: "Window and Acknowledgement Strategy in TCP" (July 1982)

This document [RFC0813] contains an early discussion of Silly Window Syndrome and its avoidance and motivates and describes the use of delayed acknowledgments.

RFC 814 U: "Name, Addresses, Ports, and Routes" (July 1982)

Suggestions and guidance for the design of tables and algorithms to keep track of various identifiers within a TCP/IP implementation are provided by this document [RFC0814].

RFC 816 U: "Fault Isolation and Recovery" (July 1982)

In this document [RFC0816], TCP's response to indications of network error conditions such as timeouts or received ICMP messages is discussed.

RFC 817 U: "Modularity and Efficiency in Protocol Implementation" (July 1982)

This document [RFC0817] contains implementation suggestions that are general and not TCP specific. However, they have been used to develop TCP implementations and describe some performance implications of the interactions between various layers in the Internet stack.

RFC 872 U: "TCP-on-a-LAN" (September 1982)

Conclusion: "The sometimes-expressed fear that using TCP on a local net is a bad idea is unfounded." [RFC0872]

RFC 896 U: "Congestion Control in IP/TCP Internetworks" (January 1984)

This document [RFC0896] contains some early experiences with congestion collapse and some initial thoughts on how to avoid it using congestion control in TCP. Furthermore, it defined an algorithm for efficient transmission of small packets that is today known as the Nagle Algorithm.

RFC 964 U: "Some Problems with the Specification of the Military Standard Transmission Control Protocol" (November 1985)

This document [RFC0964] points out several specification bugs in the US Military's MIL-STD-1778 document, which was intended as a successor to RFC 793 (see Section 2). This serves to remind us of the difficulty in specification writing (even when we work from existing documents!).

7.2. Architectural Guidelines

Some documents in this section contain architectural guidance and concerns, while others specify TCP- and congestion-control-related mechanisms that are broadly applicable and have impacts on TCP's congestion control techniques. Some of these documents are direct products of the Internet Architecture Board (IAB), giving their guidance on specific aspects of congestion control in the Internet.

RFC 1958 I: "Architectural Principles of the Internet" (June 1996)

This document [RFC1958] describes the underlying principles of the Internet architecture. It provides guidelines for network systems designs that have proven useful in the evolution of the Internet.

RFC 2914 B: "Congestion Control Principles" (September 2000)

This document [RFC2914] motivates the use of end-to-end congestion control for preventing congestion collapse and providing fairness to TCP. Later work on TCP has included several more aggressive mechanisms than Reno TCP includes, and RFC 5033 (see Section 7.4) provides additional guidance on use of such algorithms. The fundamental architectural discussion in RFC 2914 remains valid, regarding the standards process role in defining protocol aspects that are critical to performance and avoiding congestion collapse scenarios.

RFC 3360 B: "Inappropriate TCP Resets Considered Harmful" (August 2002)

This document [RFC3360] is a plea that firewall vendors not send gratuitous TCP RST (Reset) packets when unassigned TCP header bits are used. This practice prevents desirable extension and evolution of the protocol and thus is potentially harmful to the future of the Internet.

RFC 3439 I: "Some Internet Architectural Guidelines and Philosophy" (December 2002)

This document [RFC3439] updates RFC 1958 (see Section 7.2) by outlining some philosophical guidelines for architects and designers of Internet backbone networks. The document describes the Simplicity Principle, which states that complexity is the primary impediment to efficient scaling.

RFC 4774 B: "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field" (November 2006)

This document [RFC4774] discusses some of the issues in defining alternate semantics for the ECN field, and specifies requirements for a safe co-existence with routers that do not understand the defined alternate semantics.

RFC 6182 I: "Architectural Guidelines for Multipath TCP Development" (March 2011)

Abstract: "This document outlines architectural guidelines for the development of a Multipath Transport Protocol, with references to how these architectural components come together in the development of a Multipath TCP (MPTCP) (see Section 4.7). This document lists certain high-level design decisions that provide foundations for the design of the MPTCP protocol, based upon these architectural requirements" [RFC6182]

7.3. Difficult Network Environments

As the internetworking field has explored wireless, satellite, cellular telephone, and other kinds of link-layer technologies, a large body of work has built up on enhancing TCP performance for such links. The RFCs listed in this section describe some of these more challenging network environments and how TCP interacts with them.

RFC 2488 B: "Enhancing TCP Over Satellite Channels using Standard Mechanisms" (January 1999)

From abstract: "While TCP works over satellite channels there are several IETF standardized mechanisms that enable TCP to more effectively utilize the available capacity of the network path. This document outlines some of these TCP mitigations. At this time, all mitigations discussed in this document are IETF standards track mechanisms (or are compliant with IETF standards)." [RFC2488]

RFC 2757 I: "Long Thin Networks" (January 2000)

Several methods of improving TCP performance over long thin networks (i.e., networks with low bandwidth and high delay), such as geosynchronous satellite links, are discussed in this document [RFC2757]. A particular set of TCP options is developed that should work well in such environments and be safe to use in the global Internet. The implications of such environments have been further discussed in RFC 3150 (see Section 7.3) and RFC 3155 (see Section 7.3), and these documents should be preferred where there is overlap between them and RFC 2757 (see Section 7.3).

RFC 2760 I: "Ongoing TCP Research Related to Satellites" (February 2000)

This document [RFC2760] discusses the advantages and disadvantages of several different experimental means of improving TCP performance over long-delay or error-prone paths. These include T/TCP, larger initial windows, byte counting, delayed acknowledgments, slow start thresholds, NewReno and SACK-based loss recovery, FACK [MM96], ECN, various corruption-detection mechanisms, congestion avoidance changes for fairness, use of multiple parallel flows, pacing, header compression, state sharing, and ACK congestion control, filtering, and reconstruction. Although RFC 2488 (see Section 7.3) looks at standard extensions, this document focuses on more experimental means of performance enhancement.

RFC 3135 I: "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations" (June 2001)

From abstract: "This document is a survey of Performance Enhancing Proxies (PEPs) often employed to improve degraded TCP performance caused by characteristics of specific link environments, for example, in satellite, wireless WAN, and wireless LAN environments. Different types of Performance Enhancing Proxies are described as well as the mechanisms used to improve performance." [RFC3135]

RFC 3150 B: "End-to-end Performance Implications of Slow Links" (July 2001)

From abstract: "This document makes performance-related recommendations for users of network paths that traverse "very low bit-rate" links....This recommendation may be useful in any network where hosts can saturate available bandwidth, but the design space for this recommendation explicitly includes connections that traverse 56 Kb/second modem links or 4.8 Kb/

second wireless access links - both of which are widely deployed."
[RFC3150]

RFC 3155 B: "End-to-end Performance Implications of Links with Errors" (August 2001)

From abstract: "This document discusses the specific TCP mechanisms that are problematic in environments with high uncorrected error rates, and discusses what can be done to mitigate the problems without introducing intermediate devices into the connection." [RFC3155]

RFC 3366 B: "Advice to link designers on link Automatic Repeat reQuest (ARQ)" (August 2002)

From abstract: "This document provides advice to the designers of digital communication equipment and link-layer protocols employing link-layer Automatic Repeat reQuest (ARQ) techniques. This document presumes that the designers wish to support Internet protocols, but may be unfamiliar with the architecture of the Internet and with the implications of their design choices for the performance and efficiency of Internet traffic carried over their links." [RFC3366]

RFC 3449 B: "TCP Performance Implications of Network Path Asymmetry" (December 2002)

From abstract: "This document describes TCP performance problems that arise because of asymmetric effects. These problems arise in several access networks, including bandwidth-asymmetric networks and packet radio subnetworks, for different underlying reasons. However, the end result on TCP performance is the same in both cases: performance often degrades significantly because of imperfection and variability in the ACK feedback from the receiver to the sender.

The document details several mitigations to these effects, which have either been proposed or evaluated in the literature, or are currently deployed in networks." [RFC3449]

RFC 3481 B: "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks" (February 2003)

From abstract: "This document describes a profile for optimizing TCP to adapt so that it handles paths including second (2.5G) and third (3G) generation wireless networks." [RFC3481]

RFC 3819 B: "Advice for Internet Subnetwork Designers" (July 2004)

This document [RFC3819] describes how TCP performance can be negatively affected by some particular lower-layer behaviors and provides guidance in designing lower-layer networks and protocols to be amicable to TCP. RFC 3366 (see Section 7.3) specifically focuses on ARQ mechanisms, while RFC 3819 more widely covers additional aspects of the underlying layers

7.4. Guidance for Developing, Analyzing, and Evaluating TCP

Documents in this section give general guidance for developing, analyzing, and evaluating TCP. Some of the documents discuss for example the properties of congestion control protocols that are "safe" for Internet deployment, as well as how to measure the properties of congestion control mechanisms and transport protocols.

RFC 5033 B: "Specifying New Congestion Control Algorithms" (August 2007)

This document [RFC5033] considers the evaluation of suggested congestion control algorithms that differ from the principles outlined in RFC 2914 (see Section 7.2). It is useful for authors of such algorithms as well as for IETF members reviewing the associated documents.

RFC 5166 I: "Metrics for the Evaluation of Congestion Control Mechanisms" (March 2008)

This document [RFC5166] discusses metrics that needs to be considered when evaluating new or modified congestion control mechanisms for the Internet. Among others topics, the document discusses throughput, delay, loss rates, response times, fairness and robustness for challenging environments.

RFC 6077 I: "Open Research Issues in Internet Congestion Control" (January 2011)

This RFC [RFC6077] summarizes the main open problems in the domain of Internet congestion control. As a good starting point for newcomers, the document describes several new challenges that are becoming important as the network grows, as well as some issues that have been known for many years.

RFC 6181 I: "Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses" (March 2011)

This document [RFC6181] describes a threat analysis for Multipath

TCP (MPTCP) (see Section 4.7). The document discusses several types of attacks and provides recommendations for MPTCP designers how to create an MPTCP specification that is as secure as the current (single-path) TCP.

RFC 6349 I: "Framework for TCP Throughput Testing" (August 2011)

From abstract: "This document describes a practical methodology for measuring end-to-end TCP throughput in a managed IP network. The goal is to provide a better indication in regard to user experience. In this framework, TCP and IP parameters are specified to optimize TCP throughput." [RFC6349]

7.5. Implementation Advice

RFC 794 U: "PRE-EMPTION" (September 1981)

This document [RFC0794] clarifies that operating systems need to manage their limited resources, which may include TCP connection state, and that these decisions can be made with application input, but they do not need to be part of the TCP protocol specification itself.

RFC 879 U: "The TCP Maximum Segment Size and Related Topics" (November 1983)

Abstract: "This memo discusses the TCP Maximum Segment Size Option and related topics. The purpose is to clarify some aspects of TCP and its interaction with IP. This memo is a clarification to the TCP specification, and contains information that may be considered as 'advice to implementers'." [RFC0879]

RFC 1071 U: "Computing the Internet Checksum" (September 1988) (Errata)

This document [RFC1071] lists a number of implementation techniques for efficiently computing the Internet checksum (used by TCP).

RFC 1624 I: "Computation of the Internet Checksum via Incremental Update" (May 1994)

Incrementally updating the Internet checksum is useful to routers in updating IP checksums. Some middleboxes that alter TCP headers may also be able to update the TCP checksum incrementally. This document [RFC1624] expands upon the explanation of the incremental update procedure in RFC 1071 (see Section 7.5).

RFC 1936 I: "Implementing the Internet Checksum in Hardware" (April 1996)

This document [RFC1936] describes the motivation for implementing the Internet checksum in hardware, rather than in software, and provides an implementation example.

RFC 2525 I: "Known TCP Implementation Problems" (March 1999)

From abstract: "This memo catalogs a number of known TCP implementation problems. The goal is to improve conditions in the existing Internet by enhancing the quality of current TCP/IP implementations." [RFC2525]

RFC 2923 I: "TCP Problems with Path MTU Discovery" (September 2000)

From abstract: "This memo catalogs several known Transmission Control Protocol (TCP) implementation problems dealing with Path Maximum Transmission Unit Discovery (PMTUD), including the long-standing black hole problem, stretch acknowledgments (ACKs) due to confusion between Maximum Segment Size (MSS) and segment size, and MSS advertisement based on PMTU." [RFC2923]

RFC 3493 I: "Basic Socket Interface Extensions for IPv6" (February 2003)

This document [RFC3493] describes the de facto standard sockets API for programming with TCP. This API is implemented nearly ubiquitously in modern operating systems and programming languages.

RFC 6056 B: "Recommendations for Transport-Protocol Port Randomization" (December 2010)

This document [RFC6056] describes a number of simple and efficient methods for the selection of the client port number. It reduces the possibility of an attacker guessing the correct five-tuple (Protocol, Source/Destination Address, Source/Destination Port).

RFC 6191 B: "Reducing the TIME-WAIT State Using TCP timestamps" (April 2011)

This document [RFC6191] describes the usage of the TCP Timestamps option (RFC 7323, see Section 3.1) to perform heuristics to determine whether or not to allow the creation of a new incarnation of a connection that is in the TIME-WAIT state.

RFC 6429 I: "TCP Sender Clarification for Persist Condition"
(December 2011)

This document [RFC6429] clarifies the actions that a TCP can take on connections that are experiencing the Zero Window Probe (ZWP) condition.

RFC 6897 I: "Multipath TCP (MPTCP) Application Interface Considerations" (March 2013)

This document [RFC6897] characterizes the impact that Multipath TCP (MPTCP) (see Section 4.7) may have on applications. It further discusses compatibility issues of MPTCP in combination with non-MPTCP-aware applications. Finally, it describes a basic API that is a simple extension of TCP's interface for MPTCP-aware applications.

7.6. Tools and Tutorials

RFC 1180 I: "TCP/IP Tutorial" (January 1991) (Errata)

This document [RFC1180] is an extremely brief overview of the TCP/IP protocol suite as a whole. It gives some explanation as to how and where TCP fits in.

RFC 1470 I: "FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices" (June 1993)

A few of the tools that this document [RFC1470] describes are still maintained and in use today; for example, `ttcp` and `tcpdump`. However, many of the tools described do not relate specifically to TCP and are no longer used or easily available.

RFC 2398 I: "Some Testing Tools for TCP Implementors" (August 1998)

This document [RFC2398] describes a number of TCP packet generation and analysis tools. Although some of these tools are no longer readily available or widely used, for the most part they are still relevant and usable.

RFC 5783 I: "Congestion Control in the RFC Series" (February 2010)

This document [RFC5783] provides an overview of RFCs related to congestion control that have been published so far. The focus of the document is on end-host-based congestion control.

7.7. MIB Modules

The first MIB module defined for use with Simple Network Management Protocol (SNMP) was a single monolithic MIB module, called MIB-I, defined in RFC 1156. This evolved over time to the MIB-II specification in RFC 1213, which obsoletes RFC 1156. It then became apparent that having a single monolithic MIB module was not scalable, given the number and breadth of MIB data definitions that needed to be included. Thus, additional MIB modules were defined, and those parts of MIB-II that needed to evolve were split off. Eventually, the remaining parts of MIB-II were also split off, the TCP-specific part being documented in RFC 2012. RFC 2012 was obsoleted by RFC 4022, which is the primary TCP MIB document today. For current TCP implementers, RFC 4022 should be supported.

RFC 1156 S: "Management Information Base for Network Management of TCP/IP-based Internets" (May 1990)

This document [RFC1156] describes the required MIB fields for TCP implementations with minor corrections and no technical changes from RFC 1066, which it obsoletes. This is the standards track document for MIB-I.

RFC 1213 S: "Management Information Base for Network Management of TCP/IP-based Internets: MIB-II" (March 1991)

This document [RFC1213] describes the second version of the MIB in a monolithic form. It is the immediate successor of RFC 1158, with minor modifications. It obsoletes the MIB-I, defined in RFC 1156 (see Section 7.7).

RFC 2012 S: "SNMPv2 Management Information Base for the Transmission Control Protocol using SMIV2" (November 1996)

In an update to RFC 1213 (see Section 7.7), this document [RFC2012] defines the TCP MIB by splitting out the TCP-specific portions. It is now obsoleted by RFC 4022 (see Section 7.7).

RFC 2452 S: "IP Version 6 Management Information Base for the Transmission Control Protocol" (December 1998)

This document [RFC2452] augments RFC 2012 (see Section 7.7) by adding an IPv6-specific connection table. The rest of RFC 2012 holds for any IP version. RFC 2452 is now obsoleted by RFC 4022 (see Section 7.7).

Although it is a standards track document, RFC 2452 is considered a historic mistake by the MIB community, as it is based on the idea of parallel IPv4 and IPv6 structures. Although IPv6 requires new structures, the community has decided to define a single generic structure for both IPv4 and IPv6. This will aid in definition, implementation, and transition between IPv4 and IPv6.

RFC 4022 S: "Management Information Base for the Transmission Control Protocol (TCP)" (March 2005)

This document [RFC4022] obsoletes RFC 2012 (see Section 7.7) and RFC 2452 (see Section 7.7) and specifies the current standard for the TCP MIB that should be deployed.

RFC 4898 S: "TCP Extended Statistics MIB" (May 2007)

This document [RFC4898] describes extended performance statistics for TCP. They are designed to use TCP's ideal vantage point to diagnose performance problems in both the network and the application.

7.8. Case Studies

RFC 700 U: "A Protocol Experiment" (August 1974)

This document [RFC0700] presents a field report about the deployment of a very early version of TCP, the so-called INWN #39 protocol, which is originally described by Cerf and Kahn in INWG Note #39 [CK73] to use a PDP-11 line printer via the ARPANET.

RFC 889 U: "Internet Delay Experiments" (December 1983)

This document [RFC0889] is a status report about experiments concerning the TCP retransmission timeout calculation and also provides advices for implementers.

RFC 1337 I: "TIME-WAIT Assassination Hazards in TCP" (May 1992)

This document [RFC1337] points out a problem with acting on received reset segments while one is in the TIME-WAIT state. The main recommendation is that hosts in TIME-WAIT ignore resets. This recommendation might not currently be widely implemented.

RFC 2415 I: "Simulation Studies of Increased Initial TCP Window Size" (September 1998)

This document [RFC2415] presents results of some simulations using TCP initial windows greater than 1 segment. The analysis

indicates that user-perceived performance can be improved by increasing the initial window to 3 segments.

RFC 2416 I: "When TCP Starts Up With Four Packets Into Only Three Buffers" (September 1998)

This document [RFC2416] uses simulation results to clear up some concerns about using an initial window of 4 segments when the network path has less provisioning.

RFC 2884 I: "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks" (July 2000)

This document [RFC2884] describes experimental results that show some improvements to the performance of both short- and long-lived connections due to ECN.

8. Undocumented TCP Features

There are a few important implementation tactics for the TCP that have not yet been described in any RFC. Although this roadmap is primarily concerned with mapping the TCP RFCs, this section is included because an implementer needs to be aware of these important issues.

Header Prediction

Header prediction is a trick to speed up the processing of segments. Van Jacobson and Mike Karels developed the technique in the late 1980s. The basic idea is that some processing time can be saved when most of a segment's fields can be predicted from previous segments. A good description of this was sent to the TCP-IP mailing list by Van Jacobson on March 9, 1988:

"Quite a bit of the speedup comes from an algorithm that we ('we' refers to collaborator Mike Karels and myself) are calling "header prediction". The idea is that if you're in the middle of a bulk data transfer and have just seen a packet, you know what the next packet is going to look like: It will look just like the current packet with either the sequence number or ack number updated (depending on whether you're the sender or receiver). Combining this with the "Use hints" epigram from Butler Lampson's classic "Epigrams for System Designers", you start to think of the tcp state (rcv.nxt, snd.una, etc.) as "hints" about what the next packet should look like.

If you arrange those "hints" so they match the layout of a tcp packet header, it takes a single 14-byte compare to see if your prediction is correct (3 longword compares to pick up the send & ack sequence numbers, header length, flags and window, plus a short compare on the length). If the prediction is correct, there's a single test on the length to see if you're the sender or receiver followed by the appropriate processing. E.g., if the length is non-zero (you're the receiver), checksum and append the data to the socket buffer then wake any process that's sleeping on the buffer. Update rcv.nxt by the length of this packet (this updates your "prediction" of the next packet). Check if you can handle another packet the same size as the current one. If not, set one of the unused flag bits in your header prediction to guarantee that the prediction will fail on the next packet and force you to go through full protocol processing. Otherwise, you're done with this packet. So, the *total* tcp protocol processing, exclusive of checksumming, is on the order of 6 compares and an add."

Forward Acknowledgement (FACK)

FACK [MM96] includes an alternate algorithm for triggering fast retransmit [RFC5681], 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 duplicate ACK 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 duplicate ACK threshold. FACK is implemented in Linux and turned on per default.

Congestion Control for High Rate Flows

In the last decade significant research effort has been put into experimental TCP congestion control modifications for obtaining high throughput with reduced startup and recovery times. Only few RFCs have been published on some of these modifications, including HighSpeed TCP [RFC3649] (see Section 4.3), Limited Slow-Start [RFC3742] (see Section 4.3), and Quick-Start [RFC4782] (see Section 4.3), but high-rate congestion control mechanisms are still considered an open issue in congestion control research. Some other schemes have been published as Internet-Drafts, e.g. CUBIC [I-D.rhee-tcpm-cubic] (the standard TCP congestion control algorithm in Linux), Compound TCP [I-D.sridharan-tcpm-ctcp], and H-TCP [I-D.leith-tcp-htcp] or have been discussed a little by the

IETF, but much of the work in this area has not been adopted within the IETF yet, so the majority of this work is outside the RFC series and may be discussed in other products of the IRTF Internet Congestion Control Research Group (ICCRG).

9. Security Considerations

This document introduces no new security considerations. Each RFC listed in this document attempts to address the security considerations of the specification it contains.

10. IANA Considerations

This document contains no IANA considerations.

11. Acknowledgments

This document grew out of a discussion on the end2end-interest mailing list, the public list of the End-to-End Research Group of the IRTF, and continued development under the IETF's TCP Maintenance and Minor Extensions (TCPM) working group. We thank Mark Allman, Yuchung Cheng, Ted Faber, Fairhurst, Sally Floyd, Janardhan Iyengar, Reiner Ludwig, Pekka Savola, and Joe Touch for their contributions, in particular. Keith McCloghrie provided some useful notes and clarification on the various MIB-related RFCs.

12. References

12.1. Normative References

- [I-D.ietf-tcpm-1323bis]
Borman, D., Braden, R., Jacobson, V., and R. Scheffenegger, "TCP Extensions for High Performance", draft-ietf-tcpm-1323bis-21 (work in progress), April 2014.
- [I-D.ietf-tcpm-fastopen]
Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", draft-ietf-tcpm-fastopen-09 (work in progress), July 2014.
- [RFC0675] Cerf, V., Dalal, Y., and C. Sunshine, "Specification of Internet Transmission Control Program", RFC 675, December 1974.

- [RFC0700] Mader, E., Plummer, W., and R. Tomlinson, "Protocol experiment", RFC 700, August 1974.
- [RFC0721] Garlick, L., "Out-of-Band Control Signals in a Host-to-Host Protocol", RFC 721, September 1976.
- [RFC0761] Postel, J., "DoD standard Transmission Control Protocol", RFC 761, January 1980.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [RFC0794] Cerf, V., "Pre-emption", RFC 794, September 1981.
- [RFC0813] Clark, D., "Window and Acknowledgement Strategy in TCP", RFC 813, July 1982.
- [RFC0814] Clark, D., "Name, addresses, ports, and routes", RFC 814, July 1982.
- [RFC0816] Clark, D., "Fault isolation and recovery", RFC 816, July 1982.
- [RFC0817] Clark, D., "Modularity and efficiency in protocol implementation", RFC 817, July 1982.
- [RFC0872] Padlipsky, M., "TCP-on-a-LAN", RFC 872, September 1982.
- [RFC0879] Postel, J., "TCP maximum segment size and related topics", RFC 879, November 1983.
- [RFC0889] Mills, D., "Internet delay experiments", RFC 889, December 1983.
- [RFC0896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.
- [RFC0964] Sidhu, D. and T. Blumer, "Some problems with the specification of the Military Standard Transmission Control Protocol", RFC 964, November 1985.
- [RFC1071] Braden, R., Borman, D., Partridge, C., and W. Plummer, "Computing the Internet checksum", RFC 1071, September 1988.
- [RFC1078] Lottor, M., "TCP port service Multiplexer (TCPMUX)", RFC 1078, November 1988.

- [RFC1106] Fox, R., "TCP big window and NAK options", RFC 1106, June 1989.
- [RFC1110] McKenzie, A., "Problem with the TCP big window option", RFC 1110, August 1989.
- [RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [RFC1144] Jacobson, V., "Compressing TCP/IP headers for low-speed serial links", RFC 1144, February 1990.
- [RFC1146] Zweig, J. and C. Partridge, "TCP alternate checksum options", RFC 1146, March 1990.
- [RFC1156] McCloghrie, K. and M. Rose, "Management Information Base for network management of TCP/IP-based internets", RFC 1156, May 1990.
- [RFC1180] Socolofsky, T. and C. Kale, "TCP/IP tutorial", RFC 1180, January 1991.
- [RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.
- [RFC1213] McCloghrie, K. and M. Rose, "Management Information Base for Network Management of TCP/IP-based internets:MIB-II", STD 17, RFC 1213, March 1991.
- [RFC1263] O'Malley, S. and L. Peterson, "TCP Extensions Considered Harmful", RFC 1263, October 1991.
- [RFC1337] Braden, B., "TIME-WAIT Assassination Hazards in TCP", RFC 1337, May 1992.
- [RFC1379] Braden, B., "Extending TCP for Transactions -- Concepts", RFC 1379, November 1992.
- [RFC1470] Enger, R. and J. Reynolds, "FYI on a Network Management Tool Catalog: Tools for Monitoring and Debugging TCP/IP Internets and Interconnected Devices", RFC 1470, June 1993.
- [RFC1624] Rijsinghani, A., "Computation of the Internet Checksum via Incremental Update", RFC 1624, May 1994.
- [RFC1644] Braden, B., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, July 1994.

- [RFC1693] Connolly, T., Amer, P., and P. Conrad, "An Extension to TCP : Partial Order Service", RFC 1693, November 1994.
- [RFC1705] Carlson, R. and D. Ficarella, "Six Virtual Inches to the Left: The Problem with IPng", RFC 1705, October 1994.
- [RFC1936] Touch, J. and B. Parham, "Implementing the Internet Checksum in Hardware", RFC 1936, April 1996.
- [RFC1958] Carpenter, B., "Architectural Principles of the Internet", RFC 1958, June 1996.
- [RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.
- [RFC2012] McCloghrie, K., "SNMPv2 Management Information Base for the Transmission Control Protocol using SMIV2", RFC 2012, November 1996.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140, April 1997.
- [RFC2398] Parker, S. and C. Schmechel, "Some Testing Tools for TCP Implementors", RFC 2398, August 1998.
- [RFC2415] Poduri, K., "Simulation Studies of Increased Initial TCP Window Size", RFC 2415, September 1998.
- [RFC2416] Shepard, T. and C. Partridge, "When TCP Starts Up With Four Packets Into Only Three Buffers", RFC 2416, September 1998.
- [RFC2452] Daniele, M., "IP Version 6 Management Information Base for the Transmission Control Protocol", RFC 2452, December 1998.
- [RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
- [RFC2488] Allman, M., Glover, D., and L. Sanchez, "Enhancing TCP Over Satellite Channels using Standard Mechanisms", BCP 28, RFC 2488, January 1999.
- [RFC2525] Paxson, V., Dawson, S., Fenner, W., Griner, J., Heavens, I., Lahey, K., Semke, J., and B. Volz, "Known TCP

- Implementation Problems", RFC 2525, March 1999.
- [RFC2675] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms", RFC 2675, August 1999.
- [RFC2757] Montenegro, G., Dawkins, S., Kojo, M., Magret, V., and N. Vaidya, "Long Thin Networks", RFC 2757, January 2000.
- [RFC2760] Allman, M., Dawkins, S., Glover, D., Griner, J., Tran, D., Henderson, T., Heidemann, J., Touch, J., Kruse, H., Ostermann, S., Scott, K., and J. Semke, "Ongoing TCP Research Related to Satellites", RFC 2760, February 2000.
- [RFC2780] Bradner, S. and V. Paxson, "IANA Allocation Guidelines For Values In the Internet Protocol and Related Headers", BCP 37, RFC 2780, March 2000.
- [RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window Validation", RFC 2861, June 2000.
- [RFC2873] Xiao, X., Hannan, A., Paxson, V., and E. Crabbe, "TCP Processing of the IPv4 Precedence Field", RFC 2873, June 2000.
- [RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.
- [RFC2884] Hadi Salim, J. and U. Ahmed, "Performance Evaluation of Explicit Congestion Notification (ECN) in IP Networks", RFC 2884, July 2000.
- [RFC2914] Floyd, S., "Congestion Control Principles", BCP 41, RFC 2914, September 2000.
- [RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, September 2000.
- [RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.
- [RFC3124] Balakrishnan, H. and S. Seshan, "The Congestion Manager", RFC 3124, June 2001.
- [RFC3135] Border, J., Kojo, M., Griner, J., Montenegro, G., and Z. Shelby, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations", RFC 3135, June 2001.

- [RFC3150] Dawkins, S., Montenegro, G., Kojo, M., and V. Magret, "End-to-end Performance Implications of Slow Links", BCP 48, RFC 3150, July 2001.
- [RFC3155] Dawkins, S., Montenegro, G., Kojo, M., Magret, V., and N. Vaidya, "End-to-end Performance Implications of Links with Errors", BCP 50, RFC 3155, August 2001.
- [RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.
- [RFC3360] Floyd, S., "Inappropriate TCP Resets Considered Harmful", BCP 60, RFC 3360, August 2002.
- [RFC3366] Fairhurst, G. and L. Wood, "Advice to link designers on link Automatic Repeat reQuest (ARQ)", BCP 62, RFC 3366, August 2002.
- [RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.
- [RFC3439] Bush, R. and D. Meyer, "Some Internet Architectural Guidelines and Philosophy", RFC 3439, December 2002.
- [RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, December 2002.
- [RFC3465] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, February 2003.
- [RFC3481] Inamura, H., Montenegro, G., Ludwig, R., Gurtov, A., and F. Khafizov, "TCP over Second (2.5G) and Third (3G) Generation Wireless Networks", BCP 71, RFC 3481, February 2003.
- [RFC3493] Gilligan, R., Thomson, S., Bound, J., McCann, J., and W. Stevens, "Basic Socket Interface Extensions for IPv6", RFC 3493, February 2003.
- [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.
- [RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, June 2003.

- [RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows", RFC 3649, December 2003.
- [RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, February 2004.
- [RFC3742] Floyd, S., "Limited Slow-Start for TCP with Large Congestion Windows", RFC 3742, March 2004.
- [RFC3819] Karn, P., Bormann, C., Fairhurst, G., Grossman, D., Ludwig, R., Mahdavi, J., Montenegro, G., Touch, J., and L. Wood, "Advice for Internet Subnetwork Designers", BCP 89, RFC 3819, July 2004.
- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.
- [RFC4022] Raghunarayan, R., "Management Information Base for the Transmission Control Protocol (TCP)", RFC 4022, March 2005.
- [RFC4653] Bhandarkar, S., Reddy, A., Allman, M., and E. Blanton, "Improving the Robustness of TCP to Non-Congestion Events", RFC 4653, August 2006.
- [RFC4727] Fenner, B., "Experimental Values In IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers", RFC 4727, November 2006.
- [RFC4774] Floyd, S., "Specifying Alternate Semantics for the Explicit Congestion Notification (ECN) Field", BCP 124, RFC 4774, November 2006.
- [RFC4782] Floyd, S., Allman, M., Jain, A., and P. Sarolahti, "Quick-Start for TCP and IP", RFC 4782, January 2007.
- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.
- [RFC4898] Mathis, M., Heffner, J., and R. Raghunarayan, "TCP Extended Statistics MIB", RFC 4898, May 2007.
- [RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, July 2007.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common

Mitigations", RFC 4987, August 2007.

- [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion Control Algorithms", BCP 133, RFC 5033, August 2007.
- [RFC5166] Floyd, S., "Metrics for the Evaluation of Congestion Control Mechanisms", RFC 5166, March 2008.
- [RFC5461] Gont, F., "TCP's Reaction to Soft Errors", RFC 5461, February 2009.
- [RFC5482] Eggert, L. and F. Gont, "TCP User Timeout Option", RFC 5482, March 2009.
- [RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, June 2009.
- [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.
- [RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, February 2010.
- [RFC5783] Welzl, M. and W. Eddy, "Congestion Control in the RFC Series", RFC 5783, February 2010.
- [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.
- [RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.
- [RFC5926] Lebovitz, G. and E. Rescorla, "Cryptographic Algorithms for the TCP Authentication Option (TCP-AO)", RFC 5926, June 2010.
- [RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, July 2010.
- [RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's

- Robustness to Blind In-Window Attacks", RFC 5961, August 2010.
- [RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC 6013, January 2011.
- [RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, January 2011.
- [RFC6069] Zimmermann, A. and A. Hannemann, "Making TCP More Robust to Long Connectivity Disruptions (TCP-LCD)", RFC 6069, December 2010.
- [RFC6077] Papadimitriou, D., Welzl, M., Scharf, M., and B. Briscoe, "Open Research Issues in Internet Congestion Control", RFC 6077, February 2011.
- [RFC6093] Gont, F. and A. Yourtchenko, "On the Implementation of the TCP Urgent Mechanism", RFC 6093, January 2011.
- [RFC6181] Bagnulo, M., "Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6181, March 2011.
- [RFC6182] Ford, A., Raiciu, C., Handley, M., Barre, S., and J. Iyengar, "Architectural Guidelines for Multipath TCP Development", RFC 6182, March 2011.
- [RFC6191] Gont, F., "Reducing the TIME-WAIT State Using TCP Timestamps", BCP 159, RFC 6191, April 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.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.
- [RFC6349] Constantine, B., Forget, G., Geib, R., and R. Schrage,

- "Framework for TCP Throughput Testing", RFC 6349, August 2011.
- [RFC6356] Raiciu, C., Handley, M., and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols", RFC 6356, October 2011.
- [RFC6429] Bashyam, M., Jethanandani, M., and A. Ramaiah, "TCP Sender Clarification for Persist Condition", RFC 6429, December 2011.
- [RFC6528] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, February 2012.
- [RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, April 2012.
- [RFC6633] Gont, F., "Deprecation of ICMP Source Quench Messages", RFC 6633, May 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.
- [RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, January 2013.
- [RFC6846] Pelletier, G., Sandlund, K., Jonsson, L-E., and M. West, "RObust Header Compression (ROHC): A Profile for TCP/IP (ROHC-TCP)", RFC 6846, January 2013.
- [RFC6897] Scharf, M. and A. Ford, "Multipath TCP (MPTCP) Application Interface Considerations", RFC 6897, March 2013.
- [RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, April 2013.
- [RFC6937] Mathis, M., Dukkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, May 2013.
- [RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, August 2013.

12.2. Informative References

- [CK73] Cerf, V. and R. Kahn, "Towards Protocols for Internetwork Communication", IFIP/TC6.1, NIC 18764, INWG 39, September 1973.
- [Errata] "RFC Editor - RFC Errata",
<<http://www.rfc-editor.org/errata.php>>.
- [I-D.leith-tcp-htcp]
Leith, D., "H-TCP: TCP Congestion Control for High Bandwidth-Delay Product Paths", draft-leith-tcp-htcp-06 (work in progress), April 2008.
- [I-D.rhee-tcpm-cubic]
Rhee, I., Xu, L., and S. Ha, "CUBIC for Fast Long-Distance Networks", draft-rhee-tcpm-cubic-02 (work in progress), August 2008.
- [I-D.sridharan-tcpm-ctcp]
Sridharan, M., Tan, K., Bansal, D., and D. Thaler, "Compound TCP: A New TCP Congestion Control for High-Speed and Long Distance Networks", draft-sridharan-tcpm-ctcp-02 (work in progress), November 2008.
- [JK92] Jacobson, V. and M. Karels, "Congestion Avoidance and Control", This paper is a revised version of [Jac88], that includes an additional appendix. This paper has not been traditionally published, but is currently available at <ftp://ftp.ee.lbl.gov/papers/congavoid.ps.Z>. 1992.
- [Jac88] Jacobson, V., "Congestion Avoidance and Control", ACM SIGCOMM 1988 Proceedings, in ACM Computer Communication Review, 18 (4), pp. 314-329, August 1988.
- [KP87] Karn, P. and C. Partridge, "Round Trip Time Estimation", ACM SIGCOMM 1987 Proceedings, in ACM Computer Communication Review, 17 (5), pp. 2-7, August 1987.
- [MAF04] Medina, A., Allman, M., and S. Floyd, "Measuring the Evolution of Transport Protocols in the Internet", ACM Computer Communication Review, 35 (2), April 2005.
- [MM96] Mathis, M. and J. Mahdavi, "Forward Acknowledgement: Refining TCP Congestion Control", ACM SIGCOMM 1996 Proceedings, in ACM Computer Communication Review 26 (4), pp. 281-292, October 1996.

- [RFC1016] Prue, W. and J. Postel, "Something a host could do with source quench: The Source Quench Introduced Delay (SQuID)", RFC 1016, July 1987.
- [RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.
- [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.
- [RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.
- [RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.
- [RFC4341] Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control", RFC 4341, March 2006.
- [RFC6115] Li, T., "Recommendation for a Routing Architecture", RFC 6115, February 2011.
- [SCWA99] Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP Congestion Control with a Misbehaving Receiver", ACM Computer Communication Review, 29 (5), pp. 71-78, October 1999.

Authors' Addresses

Martin Duke
F5 Networks
401 Elliott Ave W
Seattle, WA 98119

Phone: 206-272-7537
Email: m.duke@f5.com

Robert Braden
USC Information Sciences Institute
Marina del Rey, CA 90292-6695

Phone: 310-448-9173
Email: braden@isi.edu

Wesley M. Eddy
MTI Systems
MS 500-ASRC; 21000 Brookpark Rd
Cleveland, OH 44135

Phone: 216-433-6682
Email: wes@mti-systems.com

Ethan Blanton

Email: elb@psg.com

Alexander Zimmermann
NetApp, Inc.
Sonnenallee 1
Kirchheim 85551
Germany

Phone: +49 89 900594712
Email: alexander.zimmermann@netapp.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 S	C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I 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 AcceECN 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 AcceECN 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, AcceECN 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 AcceECN 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.

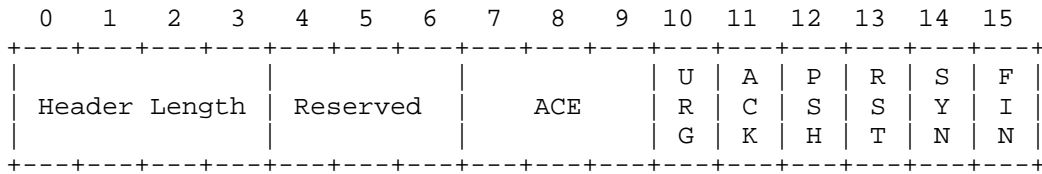


Figure 2: Definition of the ACE field within bytes 13 and 14 of the TCP Header (when AcceECN has been negotiated and SYN=0).

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 AcceECN 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 AcceECN 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 AcceECN Option

The AcceECN 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'.

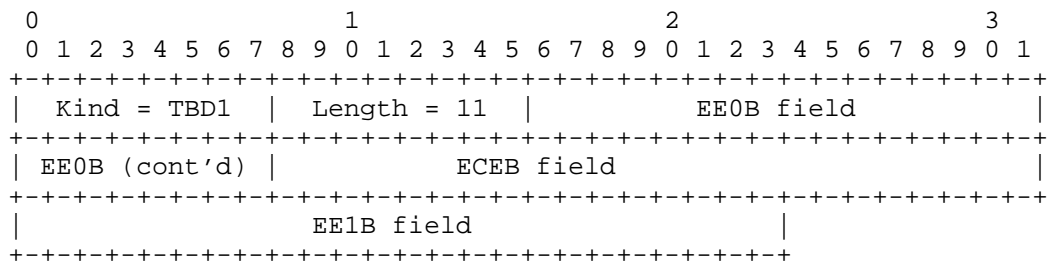


Figure 3: The AcceECN Option

The Data Receiver MUST set the Kind field to TBD1, which is registered in Section 6 as a new TCP option Kind called AcceECN. 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.ceb 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 AcceECN 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 AcceECN 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. AcceECN 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 AcceECN. Without AcceECN, the ConEx behaviour of a Data Sender would have to be more conservative than would be necessary if it had the accurate feedback of AcceECN.

- 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 `newlyAckedB`, the amount of new data that the ACK acknowledges. 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. If `newlyAckedB` is zero, to break the tie the Data Sender could use timestamps (if present) to work out `newlyAckedT`, 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
```

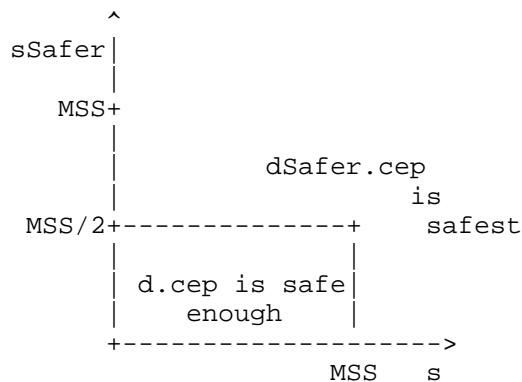
If missing acknowledgement numbers arrive later (due to reordering), Section 3.2.2 says "the Data Sender MAY attempt to neutralise the effect of any action it took based on a conservative assumption that it later found to be incorrect". To do this, the Data Sender would have to store the values of all 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 AccECN Option: It might be useful to allow an empty (Length=2) AccECN 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 AccECN in offload hardware. However, it is not known whether AccECN 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 AccECN 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
URI: <http://bobbriscoe.net/>

Mirja Kuehlewind
ETH Zurich
Gloriastrasse 35
Zurich 8092
Switzerland

EMail: 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 (ETC) 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

Richard Scheffenegger
NetApp, Inc.
Am Euro Platz 2
Vienna, 1120
Austria

Phone: +43 1 3676811 3146
Email: rs@netapp.com

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

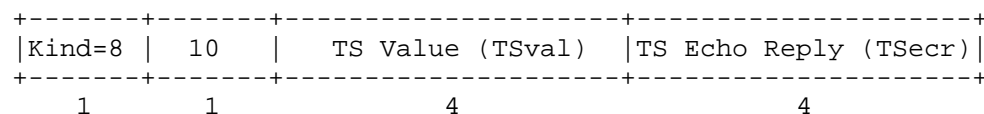


Figure 1: RFC1323 TSopt

"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 TSecr value in the initial <SYN>, or a zero TSecr 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

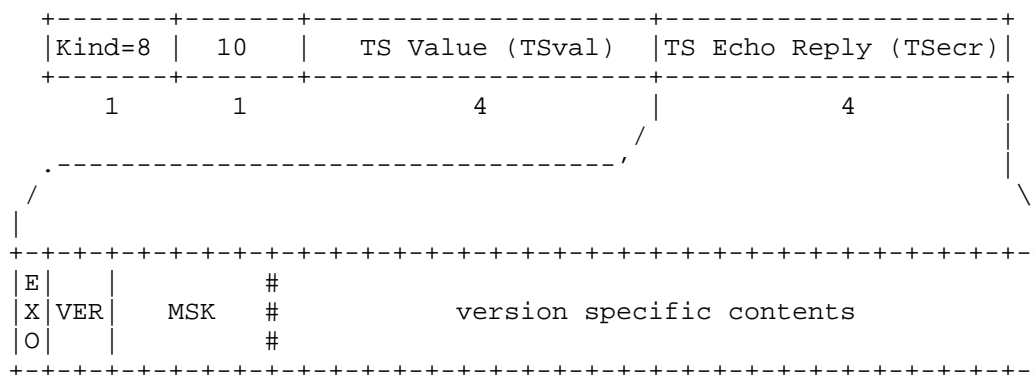


Figure 2: Timestamp Capability flags

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

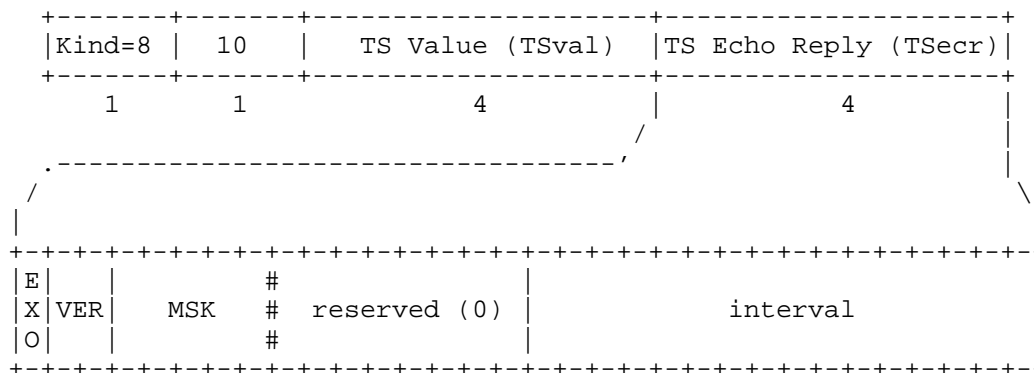


Figure 3: Timestamp Capability flags - version 0

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>>.
- [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>>.
- [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.

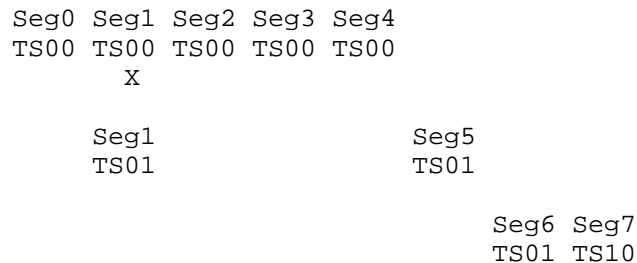


Figure 4: timestamp for spurious retransmit detection

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

TCP Maintenance and Minor Extensions (TCPM) WG
Internet-Draft
Intended status: Standards Track
Expires: May 14, 2015

A. Zimmermann
NetApp, Inc.
L. Schulte
Aalto University
C. Wolff
A. Hannemann
credativ GmbH
November 10, 2014

Detection and Quantification of Packet Reordering with TCP
draft-zimmermann-tcpm-reordering-detection-02

Abstract

This document specifies an algorithm for the detection and quantification of packet reordering for TCP. In the absence of explicit congestion notification from the network, TCP uses only packet loss as an indication of congestion. One of the signals TCP uses to determine loss is the arrival of three duplicate acknowledgments. However, this heuristic is not always correct, notably in the case when paths reorder packets. This results in degraded performance.

The algorithm for the detection and quantification of reordering in this document uses information gathered from the TCP Timestamps Option, the TCP SACK Option and its DSACK extension. When a reordering event is detected, the algorithm calculates a reordering extent by determining the number of segments the reordered segment was late with respect to its position in the sequence number space. Additionally, the algorithm computes a second reordering extent that is relative to the amount of outstanding data and thus provides a better estimation of the reordering delay when other sender state changes.

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 May 14, 2015.

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
2. Terminology	4
3. Basic Concepts	5
4. The Algorithm	5
4.1. Initialization During Connection Establishment	6
4.2. Receiving Acknowledgments	6
4.3. Receiving Acknowledgment Closing Hole	7
4.4. Receiving Duplicate Selective Acknowledgment	8
4.5. Reordering Extent Computation	8
4.6. Retransmitting Segment	9
4.7. Placeholder for Response Algorithm	9
4.8. Retransmission Timeout	9
5. Protocol Steps in Detail	9
6. Discussion of the Algorithm	12
6.1. Reasoning for the Relative Reordering Extent	12
6.2. Calculation of the Relative Reordering Extent	13
6.3. Persistent Reception of Selective Acknowledgments	13
6.4. Unreliable ACK reception	15
6.5. Packet Duplication	15
7. Related Work	16
8. IANA Considerations	17
9. Security Considerations	17
10. Acknowledgments	17
11. References	17
11.1. Normative References	17
11.2. Informative References	18

Appendix A. Changes from previous versions of the draft	20
A.1. Changes from draft-zimmermann-tcpm-reordering-detection-01	21
A.2. Changes from draft-zimmermann-tcpm-reordering-detection-00	21
Authors' Addresses	21

1. Introduction

When the Transmission Control Protocol (TCP) [RFC0793] decides that the oldest outstanding segment is lost, it performs a retransmission and changes the sending rate [RFC5681]. This occurs either when the Retransmission Timeout (RTO) timer expires for a segment [RFC6298], or when three duplicate acknowledgments (ACKs) for a segment have been received (Fast Retransmit/Fast Recovery) [RFC5681]. The assumption behind Fast Retransmit is that non-congestion events that can cause duplicate ACKs to be generated (packet duplication, packet reordering and packet corruption) are infrequent. However, a number of Internet measurement studies have shown that packet reordering is not a rare phenomenon [Pax97], [BPS99], [BS02], [ZM04], [GPL04], [JIDKT07] and has negative performance implications on TCP [BA02], [ZKFP03].

From TCP's perspective, the result of packet reordering on the forward-path is the reception of out-of-order segments by the TCP receiver. In response to every received out-of-order segment, the TCP receiver immediately sends a duplicate ACK. (Note: [RFC5681] recommends that delayed ACKs not be used when the ACK is triggered by an out-of-order segment.) The sender side, if the number of consecutively received duplicate ACKs exceeds the duplicate acknowledgment threshold (DupThresh), retransmits the first unacknowledged segment [RFC5681] and continues with a loss recovery algorithm such as NewReno [RFC6582] or the Selective Acknowledgment (SACK) based loss recovery [RFC6675]. If a segment arrives due to reordering more than three segments (the default value of DupThresh [RFC5681]) too late at the TCP receiver, the sender is not able to distinguish this reordering event from a segment loss, resulting in an unnecessary retransmission and rate reduction.

Since DupThresh is defined in segments rather than bytes [RFC5681], TCP usually quantifies packet reordering in terms of segments. Informally, the reordering extent [RFC4737] is defined as the maximum distance in segments between the reception of a reordered segment and the earliest segment received with a larger sequence number. If a segment is received in-order, its reordering extent is undefined [RFC4737].

Another approach taken by this specification quantifies the reordering extend for a packet not only through an absolute value, but also through a measure that is relative to the amount of outstanding data, in an attempt to approximate a time-based measure. The presented scheme can thereby easily be adapted to the Stream Control Transmission Protocol (SCTP) [RFC2960], since SCTP uses congestion control algorithms similar to TCP.

Overall, this document describes mechanisms to detect reordering on the forward-path during a TCP connection, and provides these samples as an input for an additional reaction algorithm.

The remainder of this document is organized as follows. Section 3 provides a high-level description of the packet reordering detection mechanisms. In Section 4, the algorithm is specified. In Section 5, each step of the algorithm is discussed in detail. Section 6 provides a discussion of several design decisions of the algorithm. Section 7 discusses related work. Section 9 discusses security concerns.

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 [RFC2119].

The reader is expected to be familiar with the TCP state variables given in [RFC0793] (SEG.SEQ, SND.UNA), [RFC5681] (FlightSize), and [RFC6675] (DupThresh, SACK scoreboard). SND.FACK (forward acknowledgment) is used to describe the highest sequence number - plus one - that has been either cumulatively or selectively acknowledged by the receiver and subsequently seen by the sender [MM96]. Further, the term 'acceptable acknowledgment' is used as defined in [RFC0793]. That is, an ACK that increases the connection's cumulative ACK point by acknowledging previously unacknowledged data. The term 'duplicate acknowledgment' is used as defined in [RFC6675], which is different from the definition of duplicate acknowledgment in [RFC5681].

This specification defines the four TCP sender states 'open', 'disorder', 'recovery', and 'loss' as follows. As long as no duplicate ACK is received and no segment is considered lost, the TCP sender is in the 'open' state. Upon the reception of the first consecutive duplicate ACK, TCP will enter the 'disorder' state. After receiving DupThresh duplicate ACKs, the TCP sender switches to the 'recovery' state and executes standard loss recovery procedures like Fast Retransmit and Fast Recovery [RFC5681]. Upon a retransmission timeout, the TCP sender enters the 'loss' state. The

'recovery' state can only be reached by a transition from the 'disorder' state, the 'loss' state can be reached from any other state.

3. Basic Concepts

The following specification depends on the TCP Timestamps option [RFC1323], the TCP Selective Acknowledgment (SACK) [RFC2018] option and the latter's Duplicate Selective Acknowledgment (DSACK) extension [RFC2883]. The reader is assumed to be familiar with the algorithms specified in these documents.

Reordering is quantified by an absolute and a relative reordering extent. If a hole in the SACK scoreboard of the TCP sender is closed either cumulatively by an acceptable ACK or selectively by a new SACK, then the absolute reordering extent is computed as the number of segments in the SACK scoreboard between the sequence number of the reordered segment and the highest selectively or cumulatively acknowledged sequence number. The relative reordering extent is then computed as the ratio between the absolute reordering extent and the FlightSize stored when entering the 'disorder' state.

If the hole that was closed in the SACK scoreboard corresponds to a segment that was not retransmitted, or if the retransmission of such a segment can be determined as a spurious retransmission by means of the Eifel detection algorithm [RFC3522], then the calculated reordering extent is immediately valid. Otherwise, if the verification of the Eifel detection algorithm has not been possible, the reordering extent is stored for a possibly subsequent DSACK. If no such DSACK is received in the next two round-trip times (RTTs), the reordering extents are discarded.

4. The Algorithm

Given that usually both the Nagle algorithm [RFC0896] [RFC1122] and the TCP Selective Acknowledgment Option [RFC2018] are enabled, a TCP sender MAY employ the following algorithm to detect and quantify the current perceived packet reordering in the network.

Without the Nagle algorithm, there is no straight forward way to accurately calculate the number of outstanding segments in the network (and, therefore, no good way to derive an appropriate reordering extent) without adding state to the TCP sender. A TCP connection that does not employ the Nagle algorithm SHOULD NOT use this methodology.

If a TCP sender implements the following algorithm, the implementation MUST follow the various specifications provided in

Sections 4.1 to 4.8. The algorithm **MUST** be executed **before** the Transmission Control Block or the SACK scoreboard have been updated by another loss recovery algorithm.

4.1. Initialization During Connection Establishment

After the completion of the TCP connection establishment, the following state variables **MUST** be initialized in the TCP transmission control block:

- (C.1) The variable `Dsack`, which indicates whether a DSACK has been received so far, and the data structure `Samples`, which stores the computed reordering extents, **MUST** be initialized as:

```
Dsack = false
Samples = []
```

- (C.2) If the TCP Timestamps option [RFC1122] has been negotiated, then the variable `Timestamps` **MUST** be activated and the data structure `Retrans_TS`, which stores the value of the `TSval` field of the retransmissions sent during Fast Recovery, **MUST** be initialized. Additionally, the data structure `Retrans_Dsack` **MAY** be used in order to detect reordering longer than RTT with Timestamps and DSACK:

```
Timestamps = true
Retrans_TS = []
Retrans_Dsack = []
```

Otherwise, the Timestamps-based detection **SHOULD** be deactivated:

```
Timestamps = false
```

4.2. Receiving Acknowledgments

For each received ACK that either a) carries SACK information, **or** b) is a full ACK that terminates the current Fast Recovery procedure, **or** c) is an acceptable ACK that is received immediately after a duplicate ACK, execute steps (A.1) to (A.4), otherwise skip to step (A.4).

- (A.1) If a) the ACK carries new SACK information, **and** b) the SACK scoreboard is empty (i.e., the TCP sender has received no SACK information from the receiver), then the TCP sender **MUST** save the amount of current outstanding data:

```
FlightSizePrev = FlightSize
```

- (A.2) If the received ACK either a) cumulatively acknowledges at most SMSS bytes, *or* b) selectively acknowledges at most SMSS bytes in the sequence number space in the SACK scoreboard, then:

The TCP sender MUST execute steps (S.1) to (S.4)

- (A.3) If a) `Timestamps == false` *and* b) the received ACK carries a DSACK option [RFC2883] and the segment identified by the DSACK option can be marked according to step (A.1) to (A.4) of [RFC3708] as a valid duplicate, then:

The TCP sender MUST execute steps (D.1) to (D.3)

- (A.4) The TCP sender MUST terminate the processing of the ACK by this algorithm and MUST continue with the default processing of the ACK.

4.3. Receiving Acknowledgment Closing Hole

- (S.1) If (a) the newly cumulatively or selectively acknowledged segment SEG is a retransmission *and* b) both equations `Dsack == false` and `Timestamps == false` hold, then the TCP sender MUST skip to step (A.4).

- (S.2) Compute the relative and absolute reordering extent `ReorExtR`, `ReorExtA`:

The TCP sender MUST execute steps (E.1) to (E.4)

- (S.3) If a) the newly acknowledged segment SEG was not retransmitted before *or* b) both equations `Timestamps == true` and `Retrans_TS[SEG.SEQ] > ACK.TSecr` hold, i.e., the ACK acknowledges the original transmission and not a retransmission, then hand over the reordering extents to an additional reaction algorithm:

The TCP sender MUST execute step (P)

- (S.4) If a) the previous step (S.3) was not executed *and* b) both equations `Dsack == true` and `Timestamps == false` hold, save the reordering extents for the newly acknowledged segment SEG for at least two RTTs:

`Samples[SEG.SEQ].ReorExtR = ReorExtR`
`Samples[SEG.SEQ].ReorExtA = ReorExtA`

- (S.5) If a) the newly acknowledged segment SEG was retransmitted before exactly once *and* b) both equations `Dsack == true` and `Timestamps == true` hold *and* c) `Retrans_TS[SEG.SEQ] == ACK.TSecr`, then save `FlightSizePrev` for this segment in order to be calculate the metrics in case a DSACK arrives, i.e. reordering delay is greater than RTT:

`Retrans_Dsack[SEG.SEQ] = FlightSizePrev`

4.4. Receiving Duplicate Selective Acknowledgment

- (D.1) If no DSACK has been received so far, the sender MUST set:

`Dsack = true`

- (D.2) If a) the previous step (D.1) was not executed *and* a reordering extent was calculated for the segment SEG identified by the DSACK option, then the TCP sender MUST restore the values of the variables `ReorExtR` and `ReorExtA` and delete the corresponding entries in the data structure:

`ReorExtR = Samples[SEG.SEQ].ReorExtR`
`ReorExtA = SAMPLES[SEG.SEQ].ReorExtA`

- (D.3) If a) step (D.1) was not executed *and* b) `FlightSizePrev` was saved in step (S.4) for the segment, then the TCP sender MUST calculate the reordering extent for the segment with the E series of steps by using the `FlightSizePrev` saved for this segment and afterwards delete the corresponding entries:

`FlightSizePrev_saved = Retrans_Dsack[SEG.SEQ]`

- (D.4) Hand the new reordering extents over to an additional reaction algorithm:

The TCP sender SHOULD execute step (P)

4.5. Reordering Extent Computation

- (E.1) `SEG.SEQ` is the sequence number of the newly cumulatively or selectively acknowledged segment SEG.
- (E.2) `SND.FACK` is the highest either cumulatively or selectively acknowledged sequence number so far plus one.
- (E.3) The TCP sender MUST compute the absolute reordering extent `ReorExtA` as

$$\text{ReorExtA} = (\text{SND.FACK} - \text{SEG.SEQ}) / \text{SMSS}$$

- (E.4) The TCP sender MUST compute the relative reordering extent `ReorExtR` as

$$\text{ReorExtR} = \text{ReorExtA} * (\text{SMSS} / \text{FlightSizePrev})$$

4.6. Retransmitting Segment

If the TCP Timestamps option [RFC1323] is used to detect packet reordering, the TCP sender must save the TCP Timestamps option of all retransmitted segments during Fast Recovery.

- (RET) If a) a segment `SEG` is retransmitted during Fast Recovery, *and* b) the equation `Timestamps = true` holds, the TCP sender MUST save the value of the `TSval` field of the retransmitted segment:

$$\text{Retrans_TS}[\text{SEG.SEQ}] = \text{SEG.TSval}$$

4.7. Placeholder for Response Algorithm

- (P) This is a placeholder for an additional reaction algorithm that takes further action using the results of this algorithm, for example, the adjustment of the `DupThresh` based on relative and absolute reordering extent `ReorExtR` and `ReorExtA`.

4.8. Retransmission Timeout

The expiration of the retransmission timer should be interpreted as an indication of a change in path characteristics, and the TCP sender should consider all saved reordering extents as outdated and delete them.

- (RTO) If an retransmission timeout (RTO) occurs, a TCP sender SHOULD reset the following variables:

```
Samples = []
Retrans_TS = []
FlightSizePrev = 0
```

5. Protocol Steps in Detail

The reception of an ACK represents the starting point for the detection scheme above. For each received SACK, DSACK or acceptable ACK that prompts the TCP sender to enter the 'disorder' state, to remain in the 'disorder' state or to leave either the 'disorder' or 'recovery' states towards the 'open' state, steps (A.1) to (A.4) are

performed. All other received ACKs are not relevant for the detection of packet reordering and can be ignored. If the TCP sender changes from the 'open' to the 'disorder' state due to the reception of a duplicate ACK (i.e., the SACK scoreboard is empty and an ACK arrives carrying new SACK information), the current amount of outstanding data, FlightSize, is stored for the subsequent calculation of the relative reordering extent (step (A.1)).

Whenever a received acceptable ACK or SACK closes a hole in the sequence number space of the SACK scoreboard either partially or completely, this is an indication of packet reordering in the network (step (A.2)). The prerequisite for an accurate quantification of the reordering is that only one segment is newly acknowledged (maximum SMSS bytes of data). If more than one segment per ACK is acknowledged, either by reordering on the reverse path or the loss of ACKs, the order in which the segments have been received by the TCP receiver is no longer accurately determinable so that in this case a reordering extent is not calculated. Finally, if the received ACK carries a DSACK option that identifies a segment that was retransmitted only once, then this is sufficient to conclude reordering (step (A.3)), so that a previously calculated reordering extent can be passed to another algorithm (steps (D.3) and (P)).

With just the information provided by the ACK field or SACK information above SND.UNA, the TCP sender is unable to distinguish whether the ACK that finally acknowledges retransmitted data (either cumulatively or selectively) was sent in response to the original segment or a retransmission of the segment. This is described as the retransmission ambiguity problem in [KP87]. Therefore, the detection and quantification of reordering depends on other means to distinguish between acknowledgments for transmission and retransmission to detect if a retransmission was spurious. If neither a DSACK has been received (Dsack == false) so far nor the TCP Timestamps option has been enabled on connection establishment (Timestamps == false) then there is no possibility for the TCP sender to identify spurious retransmissions. Hence, the processing of the received ACK by the detection algorithm must be terminated for retransmitted segments (step (S.1)). Otherwise, if the segment that corresponds to the closed hole in the sequence number space of the SACK scoreboard has not been retransmitted or the retransmission can be identified by the Eifel detection algorithm [RFC3522] as a spurious retransmission, the previously calculated reordering extent is valid (step (S.2)) and an additional reaction algorithm can be executed (steps (S.3) and (P)).

For the use of the Eifel detection it is necessary to store the TCP Timestamps option of all retransmissions sent during Fast Recovery (step (Ret)). However, if the use of the Eifel detection algorithm

is not possible (Timestamps == false), the extent of a possible reordering is stored for the possibility of a subsequent arrival of a DSACK (step (P.4)). If no such DSACK is received in the next two round-trip times, the reordering extent is discarded. Since the DSACK extension is not negotiated during connection establishment [RFC2883], the reordering extent is only stored if a DSACK was previously received for the TCP connection (DSACK == true, step (D.1)).

Regardless of whether packet reordering is detected by using the SACK-based methodology, the DSACK-based methodology, or the TCP Timestamps option, quantification of the reordering will always be done when closing a hole in the sequence number space of the SACK scoreboard (step (A.2), step (P.2)). The only difference is the time of detection, which is in the case of DSACK-based methodology at least one RTT after the time of the quantification. The absolute reordering extent ReorExtA results from the number of segments in the SACK scoreboard between the sequence number of the newly acknowledged segment and the highest either cumulatively or selectively acknowledged sequence number so far plus one (SND.FACK) (step (E.3)).

In the case that the reordering delay is longer than RTT, the reordering can not be detected by timestamps or DSACK alone, but both algorithms are needed: when a packet is retransmitted, but no reordering could be detected when it was acknowledged, then it might be possible that a DSACK arrives for this packet. Then, the reordering extent was longer than RTT and the reordering extent has to be calculated at the point in time the DSACK arrives (step D.3). Therefore, we save the FlightSizePrev for a retransmitted segment when it is acked and no reordering is detected (step S.5).

It is worth noting that the absolute reordering extent includes all segments (bytes) between the closed hole and the highest acknowledged sequence number so far, i.e., it also includes segments (bytes) that are not selectively acknowledged. The reason is that if packet reordering is considered from a temporal perspective, it is irrelevant whether there are lost segments or not. The important fact is that the lost segments have been sent after the delayed segment and before the highest acknowledged segment, which is expressed by the metric. In step (E.4), the relative reordering extent ReorExtR is then calculated by the ratio between the absolute reordering extent ReorExtA and the amount of outstanding data stored by step (A.1).

6. Discussion of the Algorithm

The focus of the following discussion is on the quantification of reordering by the relative reordering extent and to elaborate on possible sources of error, which may lead to an inaccurate detection and quantification of reordering in the network.

6.1. Reasoning for the Relative Reordering Extent

A problem that arises with the way of quantifying reordering solemnly by the absolute reordering extent is that even in the presence of constant reordering, reordering extents may vary if the transmission rate of the TCP sender changes. Therefore, by using a DupThresh that directly reflects the measured reordering extent, spurious retransmissions cannot be fully avoided.

The following example illustrates this issue. Assume a path with a reordering probability of 1%, a reordering delay of 20 ms, and a bottleneck bandwidth of 3 Mb/s. Because segments that are delayed by reordering arrive 20 ms too late, the TCP receiver can receive a maximum of $((20 * 3 * 10^3) / 8) = 7500$ bytes out-of-order before the reordered segment arrives. Hence, with a Sender Maximum Segment Size (SMSS) of 1460 bytes, the largest possible reordering extent is close to 5 segments. If the bottleneck bandwidth changes from 3 Mb/s to 4 Mb/s, the maximum reordering extent will increase to 7 segments, although the reordering delay remains constant.

This simple example shows that even with constant reordering, spurious retransmissions cannot be completely avoided if the DupThresh directly reflects the reordering extent. On the other hand, the reordering extent and the resulting DupThresh can sometimes also be much too high and do not correspond to the actual packet reordering present on the path. For example, a slow start overshoot [Hoe96], [MM96], [MSMO97] at the end of slow start might induce such a problem.

An obvious solution to the problem would be to quantify packet reordering not by calculating a reordering extent, but by using the reordering late time offset [RFC4737]. Since the reordering late time offset is not specified in segments but captures the difference between the expected and actual reception time of a reordered segment, this way of quantifying reordering is independent of the current transmission rate. Disadvantages of this approach are however a higher complexity and a worse integration into the TCP specification, since an implementation would require additional timers, whereas TCP itself is self-clocked.

6.2. Calculation of the Relative Reordering Extent

Generally, the characteristics of a relative reordering extent should be that if packet reordering on a path is constant in terms of rate and delay, the relative reordering extent should also be constant, regardless of the current transmission rate of the TCP sender. The scheme proposed in this document is to calculate the relative reordering by getting the ratio between absolute reordering (the amount of data the reordered segment was received too late) and the amount of outstanding data stored when TCP sender was entering the 'disorder' state (the maximum amount of data a reordered segment can be received too late). Therefore, the relative reordering extent expresses the portion of currently outstanding data that is selectively acknowledged before the reordered segment is cumulatively acknowledged. If the transmission rate changes, the absolute reordering extent changes as well, but together with the amount of outstanding data, and hence the relative reordering extent stays constant.

A characteristic of the calculation of the relative reordering extent on the basis of currently outstanding amount of data is that the FlightSize reflects the bandwidth-delay-product and not the transmission rate. As a consequence, the relative reordering extent is not independent of the RTT. If the RTT of the communication path changes, the amount of outstanding data changes as well, but the absolute reordering extent remains constant. Hence, the relative reordering extent adapts. In principle it is possible to design an algorithm to compute the relative reordering extent independently of the RTT and to reflect only the characteristics of packet reordering of the path. But since the calculation would be far from trivial and introducing more complexity, this is considered to be future research.

6.3. Persistent Reception of Selective Acknowledgments

Especially on paths with a high bandwidth-delay-product, it is possible that even with a minor packet reordering, several segments in a single window of data are delayed. If, in addition, the sequence numbers of those segments are widely spaced in the sequence number space and the delay caused by packet reordering is sufficiently high, this might lead to a constant reception of out-of-order data. Hence, for each received segment, regardless of whether a hole in the sequence number space of the receive window is closed or not, an ACK is sent that carries SACK information. From TCP sender's perspective, this persistent receiving of new SACK information leads to the situation that the TCP sender enters the 'disorder' state when receiving the first SACK and never leaves it

again during the connection lifetime if no segment is lost in between.

In case of the above reordering detection and quantification scheme, the persistent reception of SACK blocks causes the amount of outstanding data, which is stored when the TCP sender enters the 'disorder' state, to never be updated, since FlightSize is only saved in step (A.1) when the SACK scoreboard is empty. If the transmission rate of the TCP sender, and therefore also the maximum amount of data a reordered segment can be received too late, changes significantly during its stay in the 'disorder' state, the actual amount of reordering is not accurately determined by the relative reordering extent. A decrease of the transmission rate would result in an overestimation of the reordering extent and vice versa.

A simple solution to the problem would be to store the maximum offset in terms of sequence number space by which a reordered segment can be received too late only when entering the 'disorder' state, but individually for every potentially reordered segment, that is, for every hole in the sequence number space of the SACK scoreboard. (Note: The maximum offset in terms of sequence number space by which a reordered segment can be received too late is strictly speaking the amount of data that have been transmitted later than the reordered segment. This amount of data can only be expressed by FlightSize within the 'open' state and not within the 'disorder' state, since the cumulative ACK point may not advance).

The problem with this simple idea is that for a new hole in the SACK scoreboard, it is not possible to determine whether it is a result of packet reordering or loss, and therefore it results in increased memory usage (to store the amount of data for each hole). Additionally, the packet reordering would be inaccurately quantified if the transmission rate changes significantly for a short amount of time. For example, if the amount of outstanding data is low when entering the 'disorder' state is entered, the execution of Careful Extended Limited Transmit (as described in [I-D.zimmermann-tcpm-reordering-reaction] [RFC4653]) leads to a significant short-term change of the transmission rate. When the amount of data by which the reordering segment can be delayed is determined individually for every new hole, it leads to an overestimation of the relative reordering extent, since the maximum amount of data possible is 'artificially' reduced by Careful Extended Limited Transmit.

A solution to this problem is to store the maximum offset in terms of sequence number space by which a reordered segment can be received too late not for every segment individually (which does not guarantee an accurate calculation of the relative reordering extent) but only

sufficiently often, e.g., once per RTT. The identification of what frequency would be adequate, though, is neither trivial nor universally applicable, since a concrete solution depends on the transmission behavior of the used TCP in the 'disorder' state and whether it is more beneficial for an additional reordering response algorithm to over- or underestimate the packet reordering on the path. If, for example, TCP-aNCR [I-D.zimmermann-tcpm-reordering-reaction] is used as additional reordering response algorithm, the maximum offset in terms of sequence number space by which a reordered segment can be received too late is not only stored when entering the 'disorder' state but also updated every RTT (every cwnd worth of data transmitted without a loss) while the TCP sender stays in the 'disorder' state.

6.4. Unreliable ACK reception

ACK loss and ACK reordering are a cause for inaccuracies in samples.

6.5. Packet Duplication

Although the problem of packet duplication in today's Internet [JIDKT07], [MMMR08] is negligible, it may happen in rare cases that segments on the path to the TCP receiver are duplicated. If a segment is duplicated on the path, the first incoming segment causes the receiver to send either an acceptable ACK or a SACK, depending on whether the segment is the next expected one or not. Each subsequent identical segment then causes either a duplicate ACK or a DSACK, respectively, depending on whether the DSACK extension [RFC3708] is implemented or not.

If by a combination of packet loss and packet duplication the case occurs that a Fast Retransmit for a lost segment is duplicated on the path, the TCP sender is not able to distinguish this from packet reordering. The first received ACK closes a hole in the sequence number space of the SACK scoreboard, while the second received ACK is a valid DSACK. Although both cases are indistinguishable from a theoretical point of view, the TCP sender can take measures to ensure as far as possible that the DSACK received was not the result of packet duplication.

For this purpose, step (A.3) of the above detection method checks via the steps (A.1) to (A.4) of [RFC3708] whether the segment identified by the DSACK option is marked as a valid duplicate. Unfortunately, the steps of [RFC3708] do not check that more DSACKs have been received than retransmissions have been sent, which is a characteristic of suffering both packet reordering and packet duplication at the same time. By simply counting the received

DSACKs, for example, as additional step (A.5) in [RFC3708], this corner case can be covered as well.

7. Related Work

Because of retransmission ambiguity problem [KP87], which describes TCP sender's inability to distinguish whether the first acceptable ACK that arrives after a retransmit was sent in response to the original transmit or the retransmit, two different approaches can generally be taken to detect and quantify packet reordering. First, for transmissions (non-retransmitted segments), the detection is usually conducted by detecting a closed hole in sequence number space of the SACK scoreboard. Second, for retransmissions, the detection of packet reordering is accompanied by the detection of the spurious Fast Retransmits.

Within the IETF, several proposals have been published in the RFC series to detect and quantify packet reordering. With [RFC4737] the IPPM Working Group [IPPM] defines several metrics to evaluate whether a network path has maintained packet order on a packet-by-packet basis. [RFC4737] gives concrete, well-defined metrics, along with a methodology for applying the metric to a generic packet stream. The metric discussed in this document has a different purpose from the IPPM metrics; this document discusses a TCP specific reordering metric calculated on the TCP sender's SACK scoreboard.

Besides the IPPM work, several other proposals have been developed to detect spurious retransmissions with TCP. The Eifel detection algorithm [RFC3522] uses the TCP Timestamps option to determine whether the ACK for a given retransmit is for the original transmission or a retransmission. The F-RTO scheme [RFC5682] slightly alters TCP's sending pattern immediately following a retransmission timeout to indicate whether the retransmitted segment was needed. Finally, the DSACK-based algorithm [RFC3708] uses the TCP SACK option [RFC2018] with the DSACK extension [RFC2883] to identify unnecessary retransmissions. The mechanism for detecting packet reordering outlined in this document rely on the detection schemes of those documents (except F-RTO that only works for spurious retransmits triggered by TCP's retransmission timer), although they do not provide metrics for the reordering extent whereas the algorithm described in this document does.

RR-TCP [ZKFP03] describes a reordering detection and quantification scheme that is also based on holes in the sequence number space of the SACK scoreboard and the reception of DSACKs. For every hole in the SACK scoreboard, RR-TCP calculates a reordering extent. If the segment was retransmitted before an ACK was received, it waits for a DSACK that proves that the segment was spuriously retransmitted. The

reordering sample in such a case is the mean between the sample calculated due to the hole in the sequence number space and the sample calculated in responding to the received DSACK.

The Linux kernel [Linux] implements a reordering detection based on SACK, DSACK and TCP Timestamps option as well. The detection and quantification of non-retransmitted segments with SACK or for retransmitted segments with TCP Timestamps option operates much like the scheme described in this document, with the exception of the DSACK detection. First, Linux does not store any information (e.g., reordering extent) below the cumulative ACK point, so that DSACKs below the cumulative ACK point are ignored (for the purpose for reordering quantification). Second, Linux also does not store any information about a possible reordering event when a hole in the sequence number space of the SACK scoreboard is closed. Therefore, for a DSACK reporting a duplicate above the cumulative ACK, Linux needs to approximate the reordering on arrival of a DSACK by the distance between the DSACK and the highest selectively acknowledged segment.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

The described algorithm neither improves nor degrades the current security of TCP, since this document only detects and quantifies reordering and does not change the TCP behavior. General security considerations for SACK based loss recovery are outlined in [RFC6675].

10. Acknowledgments

The authors thank the flowgrind [Flowgrind] authors and contributors for their performance measurement tool, which give us a powerful tool to analyze TCP's congestion control and loss recovery behavior in detail.

11. References

11.1. Normative References

[I-D.zimmermann-tcpm-reordering-reaction]
Zimmermann, A., Schulte, L., Wolff, C., and A. Hannemann,
"An adaptive Robustness of TCP to Non-Congestion Events",
draft-zimmermann-tcpm-reordering-reaction-01 (work in
progress), November 2013.

- [MM96] Mathis, M. and J. Mahdavi, "Forward Acknowledgement: Refining TCP Congestion Control", ACM SIGCOMM 1996 Proceedings, in ACM Computer Communication Review 26 (4), pp. 281-292, October 1996.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [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.
- [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.
- [RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, February 2004.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

11.2. Informative References

- [BA02] Blanton, E. and M. Allman, "On Making TCP More Robust to Packet Reordering", ACM Computer Communication Review vol.32, no. 1, pp. 20-30, January 2002.
- [BPS99] Bennett, J., Partridge, C., and N. Shectman, "Packet reordering is not pathological network behavior", IEEE/ACM Transactions on Networking vol. 7, no. 6, pp. 789-798, December 1999.
- [BS02] Bellardo, J. and S. Partridge, "Measuring Packet Reordering", Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement (IMW'02) pp. 97-105, November 2002.

- [Flowgrind] "Flowgrind Home Page", <<http://www.flowgrind.net>>.
- [GPL04] Gharai, L., Perkins, C., and T. Lehman, "Packet Reordering, High Speed Networks and Transport Protocol Performance", Proceedings of the 13th IEEE International Conference on Computer Communications and Networks (ICCCN'04) pp. 73-78, October 2004.
- [Hoe96] Hoe, J., "Improving the Start-up Behavior of a Congestion Control Scheme for TCP", Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'96) pp. 270-280, August 1996.
- [IPPM] "IP Performance Metrics (IPPM) Working Group", <<http://www.ietf.org/html.charters/ippm-charter.html>>.
- [JIDKT07] Jaiswal, S., Iannaccone, G., Diot, C., Kurose, J., and D. Towsley, "Measurement and Classification of Out-of-Sequence Packets in a Tier-1 IP Backbone", IEEE/ACM Transactions on Networking vol. 15, no. 1, pp. 54-66, February 2007.
- [KP87] Karn, P. and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", ACM SIGCOMM Computer Communication Review vol. 17, no. 5, pp. 2-7, November 1987.
- [Linux] "The Linux Project", <<http://www.kernel.org>>.
- [MMMR08] Mellia, M., Meo, M., Muscariello, L., and D. Rossi, "Passive analysis of TCP anomalies", Computer Networks vol. 52, no. 14, pp. 2663-2676, October 2008.
- [MSMO97] Mathis, M., Semke, J., Mahdavi, J., and T. Ott, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm", ACM SIGCOMM Computer Communication Review vol. 27, no. 3, pp. 67-82, July 1997.
- [Pax97] Paxson, V., "End-to-End Internet Packet Dynamics", IEEE/ACM Transactions on Networking vol. 7, no.3, pp. 277-292, June 1997.
- [RFC0896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.

- [RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.
- [RFC4653] Bhandarkar, S., Reddy, A., Allman, M., and E. Blanton, "Improving the Robustness of TCP to Non-Congestion Events", RFC 4653, August 2006.
- [RFC4737] Morton, A., Ciavattone, L., Ramachandran, G., Shalunov, S., and J. Perser, "Packet Reordering Metrics", RFC 4737, November 2006.
- [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.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.
- [RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, April 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.
- [ZKFP03] Zhang, M., Karp, B., Floyd, S., and L. Peterson, "RR-TCP: A Reordering-Robust TCP with DSACK", Proceedings of the 11th IEEE International Conference on Network Protocols (ICNP'03) pp. 95-106, November 2003.
- [ZM04] Zhou, X. and P. Miegheem, "Reordering of IP Packets in Internet", Lecture Notes in Computer Science vol. 3015, pp. 237-246, April 2004.

Appendix A. Changes from previous versions of the draft

This appendix should be removed by the RFC Editor before publishing this document as an RFC.

A.1. Changes from draft-zimmermann-tcpm-reordering-detection-01

- o Moved reasoning for relative reordering extent to discussion
- o Extended algorithm for calculation of reordering extents greater than RTT (steps C.2, S.5 and D.3)
- o Remove reverse-path reordering from intro

A.2. Changes from draft-zimmermann-tcpm-reordering-detection-00

- o Improved the wording throughout the document.
- o Replaced and updated some references.

Authors' Addresses

Alexander Zimmermann
NetApp, Inc.
Sonnenallee 1
Kirchheim 85551
Germany

Phone: +49 89 900594712
Email: alexander.zimmermann@netapp.com

Lennart Schulte
Aalto University
Otakaari 5 A
Espoo 02150
Finland

Phone: +358 50 4355233
Email: lennart.schulte@aalto.fi

Carsten Wolff
credativ GmbH
Hohenzollernstrasse 133
Moenchengladbach 41061
Germany

Phone: +49 2161 4643 182
Email: carsten.wolff@credativ.de

Arnd Hannemann
credativ GmbH
Hohenzollernstrasse 133
Moenchengladbach 41061
Germany

Phone: +49 2161 4643 134
Email: arnd.hannemann@credativ.de

TCP Maintenance and Minor Extensions (TCPM) WG
Internet-Draft
Obsoletes: 4653 (if approved)
Intended status: Experimental
Expires: May 14, 2015

A. Zimmermann
NetApp, Inc.
L. Schulte
Aalto University
C. Wolff
A. Hannemann
credativ GmbH
November 10, 2014

Making TCP Adaptively Robust to Non-Congestion Events
draft-zimmermann-tcpm-reordering-reaction-02

Abstract

This document specifies an adaptive Non-Congestion Robustness (aNCR) mechanism for TCP. In the absence of explicit congestion notification from the network, TCP uses only packet loss as an indication of congestion. One of the signals TCP uses to determine loss is the arrival of three duplicate acknowledgments. However, this heuristic is not always correct, notably in the case when paths reorder packets. This results in degraded performance.

TCP-aNCR is designed to mitigate this performance degradation by adaptively increasing the number of duplicate acknowledgments required to trigger loss recovery, based on the current state of the connection, in an effort to better disambiguate true segment loss from segment reordering. This document specifies the changes to TCP and TCP-NCR (on which this specification is build on) and discusses the costs and benefits of these modifications.

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 May 14, 2015.

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
2. Terminology	6
3. Basic Concept	7
4. Appropriate Detection and Quantification Algorithms	7
5. The TCP-aNCR Algorithm	8
5.1. Initialization during Connection Establishment	8
5.2. Initializing Extended Limited Transmit	9
5.3. Executing Extended Limited Transmit	10
5.4. Terminating Extended Limited Transmit	11
5.5. Entering Loss Recovery	13
5.6. Reordering Extent	13
5.7. Retransmission Timeout	13
6. Protocol Steps in Detail	14
7. Discussion of TCP-aNCR	16
7.1. Variable Duplicate Acknowledgment Threshold	16
7.2. Relative Reordering Extent	17
7.3. Reordering during Slow Start	18
7.4. Preventing Bursts	18
7.5. Persistent receiving of Selective Acknowledgments	19
8. Interoperability Issues	21
8.1. Early Retransmit	21
8.2. Congestion Window Validation	21
8.3. Reactive Response to Packet Reordering	22
8.4. Buffer Auto-Tuning	22
9. Related Work	23
10. IANA Considerations	24
11. Security Considerations	24
12. Acknowledgments	25
13. References	25
13.1. Normative References	25
13.2. Informative References	26

Appendix A. Changes from previous versions of the draft	28
A.1. Changes from draft-zimmermann-tcpm-reordering-reaction-01	28
A.2. Changes from draft-zimmermann-tcpm-reordering-reaction-00	28
Authors' Addresses	28

1. Introduction

One strength of the Transmission Control Protocol (TCP) [RFC0793] lies in its ability to adjust its sending rate according to the perceived congestion in the network [RFC5681]. In the absence of explicit notification of congestion from the network, TCP uses segment loss as an indication of congestion (i.e., assuming queue overflow). A TCP receiver sends cumulative acknowledgments (ACKs) indicating the next sequence number expected from the sender for arriving segments [RFC0793]. When segments arrive out of order, duplicate ACKs are generated. As specified in [RFC5681], a TCP sender uses the arrival of three duplicate ACKs as an indication of segment loss. The TCP sender retransmits the segment assumed lost and reduces the sending rate, based on the assumption that the loss was caused by resource contention on the path. The TCP sender does not assume loss on the first or second duplicate ACK, but waits for three duplicate ACKs to account for minor packet reordering. However, the use of this constant threshold of duplicate ACKs leads to performance degradation if the extent of the packet reordering in the network increases [RFC4653].

Whenever interoperability with the TCP congestion control and loss recovery standard [RFC5681] is a prerequisite, increasing the duplicate acknowledgment threshold (DupThresh) is the method of choice to a priori prevent any negative impact - in particular, a spurious Fast Retransmit and Fast Recovery phase - that packet reordering has on TCP. However, this procedure also delays a Fast Retransmit by increasing the DupThresh, and therefore has costs and risks, too. According to [ZKFP03], these are: (1) a delayed response to congestion in the network, (2) a potential expiration of the retransmission timer, and (3) a significant increase in the end-to-end delay for lost segments.

In the current TCP standard, congestion control and loss recovery are tightly coupled: when the oldest outstanding segment is declared lost, a retransmission is triggered, and the sending rate is reduced on the assumption that the loss is due to resource contention [RFC5681]. Therefore, any change to DupThresh causes not only a change to the loss recovery, but also to the congestion control response. TCP-NCR [RFC4653] addresses this problem by defining two extensions to TCP's Limited Transmit [RFC3042] scheme: Careful and Aggressive Extended Limited Transmit.

The first variant of the two, Careful Limited Transmit, sends one previously unsent segment in response to duplicate acknowledgments for every two segments that are known to have left the network. This effectively halves the sending rate, since normal TCP operation sends one new segment for every segment that has left the network. Further, the halving starts immediately and is not delayed until a retransmission is triggered. In the case of packet reordering (i.e., not segment loss), TCP-NCR restores the congestion control state to its previous state after the event.

The second variant, Aggressive Limited Transmit, transmits one previously unsent data segment in response to duplicate acknowledgments for every segment known to have left the network. With this variant, while waiting to disambiguate the loss from a reordering event, ACK-clocked transmission continues at roughly the same rate as before the event started. Retransmission and the sending rate reduction happen per [RFC5681] [RFC6675], albeit after a delay caused by the increased DupThresh. Although this approach delays legitimate rate reductions (possibly slightly, and temporarily aggravating overall congestion on the network), the scheme has the advantage of not reducing the transmission rate in the face of packet reordering.

A basic requirement for preventing an avoidable expiration of the retransmission timer is to generally ensure that an increased DupThresh can potentially be reached in time so that Fast Retransmit is triggered and Fast Recovery is completed before the RTO expires. Simply increasing DupThresh before retransmitting a segment can make TCP brittle to packet or ACK loss, since such loss reduces the number of duplicate ACKs that will arrive at the sender from the receiver. For instance, if cwnd is 10 segments and one segment is lost, a DupThresh of 10 will never be met, because duplicate ACKs corresponding to at most 9 segments will arrive at the sender. To mitigate this issue, the TCP-NCR [RFC4653] modification makes two fundamental changes to the way [RFC5681] [RFC6675] currently operates.

First, as mentioned above, TCP-NCR [RFC4653] extends TCP's Limited Transmit [RFC3042] scheme to allow for the sending of new data segment while the TCP sender stays in the 'disorder' state and disambiguate loss and reordering. This new data serves to increase the likelihood that enough duplicate ACKs arrive at the sender to trigger loss recovery, if it is appropriate. Second, DupThresh is increased from the current fixed value of three [RFC5681] to a value indicating that approximately a congestion window's worth of data has left the network. Since cwnd represents the amount of data a TCP sender can transmit in one round-trip time (RTT), this corresponds to

approximately the largest amount of time a TCP sender can wait before the costly retransmission timeout may be triggered.

Of vital importance is that TCP-NCR [RFC4653] holds DupThresh not constant, but dynamically adjusts it on each SACK to the current amount of outstanding data, which depends not only on the congestion window, but also on the receiver's advertised window. Thus, it is guaranteed that the outstanding data generates a sufficient number of duplicate ACKs for reaching DupThresh and a transition to the 'recovery' state. This is important in cases where there is no new data available to send.

Regarding the problem of packet reordering, TCP-NCR's [RFC4653] decision of waiting to receive notice that cwnd bytes have left the network before deciding whether the root cause is loss or reordering is essentially a trade-off between making the best decision regarding the cause of the duplicate ACKs and responsiveness, and represents a good compromise between avoiding spurious Fast Retransmits and avoiding unnecessary RTOs. On the other hand, if there is no visible packet reordering on the network path - which today is the rule and not the exception - or the delay caused by the reordering is very low, delaying Fast Retransmit is unnecessary in the case of congestion, and data is delivered to the application up to one RTT later. Especially for delay-sensitive applications, such as a terminal session over SSH, this is generally undesirable. By dynamically adapting DupThresh not only to the amount of outstanding data but also to the perceived packet reordering on the network path, this issue can be offset. This is the key idea behind the TCP-aNCR algorithm.

This document specifies a set of TCP modifications to provide an adaptive Non-Congestion Robustness (aNCR) mechanism for TCP. The TCP-aNCR modifications lend themselves to incremental deployment. Only the TCP implementation on the sender side requires modification. The changes themselves are modest. TCP-aNCR is built on top of the TCP Selective Acknowledgments Option [RFC2018] and the SACK-based loss recovery scheme given in [RFC6675] and represents an enhancement of the original TCP-NCR mechanism [RFC4653]. Currently, TCP-aNCR is an independent approach of making TCP more robust to packet reordering. It is not clear if upcoming versions of this draft TCP-aNCR will obsolete TCP-NCR or not.

It should be noted that the TCP-aNCR algorithm in this document could be easily adapted to the Stream Control Transmission Protocol (SCTP) [RFC2960], since SCTP uses congestion control algorithms similar to TCP (and thus has the same reordering robustness issues).

The remainder of this document is organized as follows. Section 3 provides a high-level description of the TCP-aNCR mechanism. Section 4 defines TCP-aNCR's requirements for an appropriate detection and quantification algorithm. Section 5 specifies the TCP-aNCR algorithm and Section 6 discusses each step of the algorithm in detail. Section 7 provides a discussion of several design decisions behind TCP-aNCR. Section 8 discusses interoperability issues related to introducing TCP-aNCR. Finally, related work is presented in Section 9 and security concerns in Section 11.

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 [RFC2119].

The reader is expected to be familiar with the TCP state variables described in [RFC0793] (SND.NXT), [RFC5681] (cwnd, rwnd, ssthresh, FlightSize, IW), [RFC6675] (pipe, DupThresh, SACK scoreboard), and [RFC6582] (recover). Further, the term 'acceptable acknowledgment' is used as defined in [RFC0793]. That is, an ACK that increases the connection's cumulative ACK point by acknowledging previously unacknowledged data. The term 'duplicate acknowledgment' is used as defined in [RFC6675], which is different from the definition of duplicate acknowledgment in [RFC5681].

This specification defines the four TCP sender states 'open', 'disorder', 'recovery', and 'loss' as follows. As long as no duplicate ACK is received and no segment is considered lost, the TCP sender is in the 'open' state. Upon the reception of the first consecutive duplicate ACK, TCP will enter the 'disorder' state. After receiving DupThresh duplicate ACKs, the TCP sender switches to the 'recovery' state and executes standard loss recovery procedures like Fast Retransmit and Fast Recovery [RFC5681]. Upon a retransmission timeout, the TCP sender enters the 'loss' state. The 'recovery' state can only be reached by a transition from the 'disorder' state, the 'loss' state can be reached from any other state.

The following specification depends on the standard TCP congestion control and loss recovery algorithms and the SACK-based loss recovery scheme given in [RFC5681], respectively [RFC6675]. The algorithm presents an enhancement of TCP-NCR [RFC4653]. The reader is assumed to be familiar with the algorithms specified in these documents.

3. Basic Concept

The general idea behind the TCP-aNCR algorithm is to extend the TCP-NCR algorithm [RFC4653], so that - based on an appropriate packet reordering detection and quantification algorithm (see Section 4) - TCP congestion control and loss recovery [RFC5681] is adaptively adjusted to the actual perceived packet reordering on the network path.

TCP-NCR [RFC4653] increases DupThresh from the current fixed value of three duplicate ACKs [RFC5681] to approximately until a congestion window of data has left the network. Since cwnd represents the amount of data a TCP sender can transmit in one RTT, the choice to trigger a retransmission only after a cwnd's worth of data is known to have left the network represents roughly the largest amount of time a TCP sender can wait before the RTO may be triggered. The approach chosen in TCP-aNCR is to take TCP-NCR's DupThresh as an upper bound for an adjustment of the DupThresh that is adaptive to the actual packet reordering on the network path.

Using TCP-NCR's DupThresh as an upper bound decouples the avoidance of spurious Fast Retransmits from the avoidance of unnecessary retransmission timeouts. Therefore, the adaptive adjustment of the DupThresh to current perceived packet reordering can be conducted without taking any retransmission timeout avoidance strategy into account. This independence allows TCP-aNCR to quickly respond to perceived packet reordering by setting its DupThresh so that it always corresponds to the minimum of the maximum possible (TCP-NCR's DupThresh) and the maximum measured reordering extent since the last RTO. The reordering extent used by TCP-aNCR is by itself not a static absolute reordering extent, but a relative reordering extent (see Section 4).

4. Appropriate Detection and Quantification Algorithms

If the TCP-aNCR algorithm is implemented at the TCP sender, it MUST be implemented together with an appropriate packet reordering detection and quantification algorithm that is specified in a standards track or experimental RFC.

Designers of reordering detection algorithms who want their algorithms to work together with the TCP-aNCR algorithm SHOULD reuse the variable 'ReorExtR' (relative reordering extent) with the semantics and defined values specified in [I-D.zimmermann-tcpm-reordering-detection]. A 'ReorExtR' given by the detection algorithm holds a value ranging from 0 to 1 which holds the new measured reordering sample as a fraction of the data in

flight. TCP-aNCR then saves this new fraction if it is greater than the current value.

5. The TCP-aNCR Algorithm

When both the Nagle algorithm [RFC0896] [RFC1122] and the TCP Selective Acknowledgment Option [RFC2018] are enabled for a connection, a TCP sender MAY employ the following TCP-aNCR algorithm to dynamically adapt TCP's congestion control and loss recovery [RFC5681] to the currently perceived packet reordering on the network path.

Without the Nagle algorithm, there is no straightforward way to accurately calculate the number of outstanding segments in the network (and, therefore, no good way to derive an appropriate DupThresh) without adding state to the TCP sender. A TCP connection that does not use the Nagle algorithm SHOULD NOT use TCP-aNCR. The adaptation of TCP-aNCR to an implementation that carefully tracks the sequence numbers transmitted in each segment is considered future work.

A necessary prerequisite for TCP-aNCR's adaptability is that a TCP sender has enabled an appropriate detection and quantification algorithm that complies with the requirements defined in Section 4. If such an algorithm is either non-existent or not used, the behavior of TCP-aNCR is completely analogous to the TCP-NCR algorithm as defined in [RFC4653]. If a TCP sender does implement TCP-aNCR, the implementation MUST follow the various specifications provided in Sections 5.1 to 5.7.

5.1. Initialization during Connection Establishment

After the completion of the TCP connection establishment, the following state constants and variables MUST be initialized in the TCP transmission control block for the given TCP connection:

- (C.1) Depending on which variant of Extended Limited Transmit should be executed, the constant LT_F MUST be initialized as follows.
For Careful Extended Limited Transmit:

LT_F = 2/3

For Aggressive Extended Limited Transmit:

LT_F = 1/2

This constant reflects the fraction of outstanding data (including data sent during Extended Limited Transmit) that

must be SACKed before a retransmission is at the latest triggered.

- (C.2) If TCP-aNCR should adaptively adjust the DupThresh to the current perceived packet reordering on the network path, then the variable 'ReorExtR', which stores the maximum relative reordering extent, MUST initialized as:

ReorExtR = 0

Otherwise the dynamically adaptation of TCP-aNCR SHOULD be disabled by setting

ReorExtR = -1

A relative reordering extent of 0 results in the standard DupThresh of three duplicate ACKs, as defined in [RFC5681]. A fixed relative reordering extent of -1 results in the TCP-NCR behavior from [RFC4653].

5.2. Initializing Extended Limited Transmit

If the SACK scoreboard is empty upon the receipt of a duplicate ACK (i.e., the TCP sender has received no SACK information from the receiver), a TCP sender MUST enter Extended Limited Transmit by initialize the following five state variables in the TCP Transmission Control Block:

- (I.1) The TCP sender MUST save the current outstanding data:

FlightSizePrev = FlightSize

- (I.2) The TCP sender MUST save the highest sequence number transmitted so far:

recover = SND.NXT - 1

Note: The state variable 'recover' from [RFC6582] can be reused, since NewReno TCP uses 'recover' at the initialization of a loss recovery procedure, whereas TCP-aNCR uses 'recover' *before* loss recovery.

- (I.3) The TCP sender MUST initialize the variable 'skipped' that tracks the number of segments for which an ACK does not trigger a transmission during Careful Limited Transmit:

skipped = 0

During Aggressive Limited Transmit, 'skipped' is not used.

- (I.4) The TCP sender MUST set DupThresh based on the current FlightSize:

$$\text{DupThresh} = \max (\text{LT_F} * (\text{FlightSize} / \text{SMSS}), 3)$$

The lower bound of DupThresh = 3 is kept from [RFC5681] [RFC6675].

- (I.5) If (ReorExtR != -1) holds, then the TCP sender MUST set DupThresh based on the relative reordering extent 'ReorExtR':

$$\begin{aligned} \text{DupThresh} = \\ \max (\min (\text{DupThresh}, \\ \text{ReorExtR} * (\text{FlightSizePrev} / \text{SMSS})), 3) \end{aligned}$$

In addition to the above steps, the incoming ACK MUST be processed with the (E) series of steps in Section 5.3.

5.3. Executing Extended Limited Transmit

On each ACK that a) arrives after TCP-aNCR has entered the Extended Limited Transmit phase (as outlined in Section 5.2) *and* b) carries new SACK information, *and* c) does *not* advance the cumulative ACK point, the TCP sender MUST use the following procedure.

- (E.1) The TCP sender MUST update the SACK scoreboard and uses the SetPipe() procedure from [RFC6675] to set the 'pipe' variable (which represents the number of bytes still considered "in the network"). Note: the current value of DupThresh MUST be used by SetPipe() to produce an accurate assessment of the amount of data still considered in the network.
- (E.2) The TCP sender MUST initialize the variable 'burst' that tracks the number of segments that can at most be sent per ACK to the size of the Initial Window (IW) [RFC5681]:

$$\text{burst} = \text{IW}$$

- (E.3) If a) (cwnd - pipe - skipped >= 1 * SMSS) holds, *and* b) the receive window (rwnd) allows to send SMSS bytes of previously unsent data, *and* c) there are SMSS bytes of previously unsent data available for transmission, then the TCP sender MUST transmit one segment of SMSS bytes. Otherwise, the TCP sender MUST skip to step (E.7).

- (E.4) The TCP sender MUST increment 'pipe' by SMSS bytes and MUST decrement 'burst' by SMSS bytes to reflect the newly transmitted segment:

```
pipe = pipe + SMSS
burst = burst - SMSS
```

- (E.5) If Careful Limited Transmit is used, 'skipped' MUST be incremented by SMSS bytes to ensure that the next SMSS bytes of SACKed data processed do not trigger a Limited Transmit transmission.

```
skipped = skipped + SMSS
```

- (E.6) If (burst > 0) holds, the TCP sender MUST return to step (E.3) to ensure that as many bytes as appropriate are transmitted. Otherwise, if more than IW bytes were SACKed by a single ACK, the TCP sender MUST skip to step (E.7). The additional amount of data becomes available again by the next received duplicate ACK and the re-execution of SetPipe().

- (E.7) The TCP sender MUST save the maximum amount of data that is considered to have been in the network during the last RTT:

```
pipe_max = max (pipe, pipe_max)
```

- (E.8) The TCP sender MUST set DupThresh based on the current FlightSize:

```
DupThresh = max (LT_F * (FlightSize / SMSS), 3)
```

The lower bound of DupThresh = 3 is kept from [RFC5681] [RFC6675].

- (E.9) If (ReorExtR != -1) holds, then the TCP sender MUST set DupThresh based on the relative reordering extent 'ReorExtR':

```
DupThresh =
    max (min (DupThresh,
              ReorExtR * (FlightSizePrev / SMSS)), 3)
```

5.4. Terminating Extended Limited Transmit

On the receipt of a duplicate ACK that a) arrives after TCP-aNCR has entered the Extended Limited Transmit phase (as outlined in Section 5.2) *and* b) advances the cumulative ACK point, the TCP sender MUST use the following procedure.

The arrival of an acceptable ACK that advances the cumulative ACK point while in Extended Limited Transmit, but before loss recovery is triggered, signals that a series of duplicate ACKs was caused by reordering and not congestion. Therefore, Extended Limited Transmit will be either terminated or re-entered.

(T.1) If the received ACK extends not only the cumulative ACK point, but *also* carries new SACK information (i.e., the ACK is both an acceptable ACK and a duplicate ACK), the TCP sender *MUST* restart Extended Limited Transmit and *MUST* go to step (T.2). Otherwise, the TCP sender *MUST* terminate it and *MUST* skip to step (T.3).

(T.2) If the Cumulative Acknowledgment field of the received ACK covers more than 'recover' (i.e., $\text{SEG.ACK} > \text{recover}$), Extended Limited Transmit has transmitted one cwnd worth of data without any losses and the TCP sender *MUST* update the following state variables by

```
FlightSizePrev = pipe_max
pipe_max = 0
```

and *MUST* go to step (I.2) to re-start Extended Limited Transmit. Otherwise if $(\text{SEG.ACK} \leq \text{recover})$ holds, the TCP sender *MUST* go to step (I.3). This ensures that in the event of a loss the cwnd reduction is based on a current value of FlightSizePrev.

The following steps are executed only if the received ACK does *not* carry SACK information. Extended Limited Transmit will be terminated.

(T.3) A TCP sender *MUST* set ssthresh to:

```
ssthresh = max (cwnd, ssthresh)
```

This step provides TCP-aNCR with a sense of "history". If the next step (T.4) reduces the congestion window, this step ensures that TCP-aNCR will slow-start back to the operating point that was in effect before Extended Limited Transmit.

(T.4) A TCP sender *MUST* reset cwnd to:

```
cwnd = FlightSize + SMSS
```

This step ensures that cwnd is not significantly larger than the amount of data outstanding, a situation that would cause a line rate burst.

- (T.5) A TCP is now permitted to transmit previously unsent data as allowed by `cwnd`, `FlightSize`, application data availability, and the receiver's advertised window.

5.5. Entering Loss Recovery

The receipt of an ACK that results in deeming the oldest outstanding segment is lost via the algorithms in [RFC6675] terminates Extended Limited Transmit and initializes the loss recovery according to [RFC6675]. One slight change to either [RFC6675], or, if Proportional Rate Reduction (PRR) algorithm is used, to [RFC6937] MUST be made, however.

- (Ret) If the PRR algorithm is used to calculate how many bytes should be sent in response to each ACK, the initialization of 'RecoverFS' in Section 3 of [RFC6937] MUST be changed to:

`RecoverFS = FlightSizePrev`

Otherwise, if the standard Fast Recovery algorithm is used, step (4.2) of [RFC6675] MUST be changed in Section 5 to:

`ssthresh = cwnd = (FlightSizePrev / 2)`

This change ensures that the congestion control modifications are made with respect to the amount of data in the network before `FlightSize` was increased by Extended Limited Transmit.

Once the algorithm in [RFC6675] takes over from Extended Limited Transmit, the `DupThresh` value MUST be held constant until the loss recovery phase terminates.

5.6. Reordering Extent

Whenever the additional detection and quantification algorithm (see Section 4) detects and quantifies a new reordering event, the TCP sender MUST update the state variable 'ReorExtR'.

- (Ext) Let 'ReorExtR_New' the newly determined relative reordering extent:

`ReorExtR = min (max (ReorExtR, ReorExtR_New), 1)`

5.7. Retransmission Timeout

The expiration of the retransmission timer SHOULD be interpreted as an indication of a path characteristics change, and the TCP sender SHOULD reset `DupThresh` to the default value of three.

(RTO) If an RTO occurs and ($\text{ReorExtR} \neq -1$) (i.e. TCP-aNCR is used and not TCP-NCR), then a TCP sender SHOULD reset 'ReorExtR':

$\text{ReorExtR} = 0$

6. Protocol Steps in Detail

Upon the receipt of the first duplicate ACK in the 'open' state (the SACK scoreboard is empty), the TCP sender starts to execute TCP-aNCR by entering the 'disorder' state and the initialization of Extended Limited Transmit. First, the TCP sender saves the current amount of outstanding data as well as the highest sequence number transmitted so far ($\text{SND.NXT} - 1$) (steps (I.1) and (I.2)). In addition, if the TCP connection uses the careful variant of the Extended Careful Limited Transmit (step (C.1)), the 'skipped' variable, which tracks the number of segments for which an ACK does not trigger a transmission during Careful Limited Transmit, is initialized with zero (step (I.3)). The last step during the initialization is the determination of DupThresh. Depending on whether TCP-aNCR has been configured during the connection establishment to adaptively adjust to the currently perceived packet reordering on the path (step (C.2)), DupThresh is either determined exclusively based on the current FlightSize (as TCP-NCR [RFC4653] does) or, in addition, also based on the relative extent reordering (steps (I.4) and (I.5)).

Depending on which variant of Extended Limited Transmit should be executed, the constant LT_F must be set accordingly (step (C.1)). This constant reflects the fraction of outstanding data (including data sent during Extended Limited Transmit) that must be SACKed before a retransmission is triggered at the latest (which is the case when a DupThresh that is based on relative reordering extent is larger than TCP-NCR's DupThresh). Since Aggressive Limited Transmit sends a new segment for every segment known to have left the network, a total of approximately cwnd segments will be sent, and therefore ideally a total of approximately $2 \cdot \text{cwnd}$ segments will be outstanding when a retransmission is finally triggered. DupThresh is then set to $\text{LT_F} = 1/2$ of $2 \cdot \text{cwnd}$ (or about 1 RTT's worth of data) (see step (I.4)). The factor is different for Careful Limited Transmit, because the sender only transmits one new segment for every two segments that are SACKed and therefore will ideally have a total of maximum of $1.5 \cdot \text{cwnd}$ segments outstanding when the retransmission is triggered. Hence, the required threshold is $\text{LT_F} = 2/3$ of $1.5 \cdot \text{cwnd}$ to delay the retransmission by roughly 1 RTT.

For each duplicate ACK received in the 'disorder' state, which is not an acceptable ACK, i.e., it carries new SACK information, but does not advance the cumulative ACK point, Extended Limited Transmit is executed. First, the SACK scoreboard is updated and based on the

current value of DupThresh, the amount of outstanding data (step (E.1)). Furthermore, the state variable 'burst' that indicates the number of segments that can be sent at most for of each received ACK is initialized to the size of the initial window [RFC6928] (step E.2)). If more than IW bytes were SACKed by a single ACK, the additional amount of data becomes available again by the next received duplicate ACK and the re-execution of SetPipe() (step (E.1)).

Next, if new data is available for transmission and both the congestion window and the receiver window allow to send SMSS bytes of previously unsent data, a segment of SMSS bytes is sent (step (E.3)). Subsequently, the corresponding state variables 'pipe', 'burst' and - optionally - 'skipped' are updated (steps (E.4) and (E.5)). If, due to the current size of the congestion and receiver windows (step (E.2)), due to the current value of 'burst' (step (E.5)), no further segment may be sent, the processing of the ACK is terminated. Provided that the amount of data that is currently considered to be in the network is greater than the previously stored one, this new value is stored for later use (step (E.7)). Finally, to take into account the new data sent, DupThresh is updated (steps (E.6) and (E.7)).

The arrival of an acceptable ACK in the 'disorder' state that advances the cumulative ACK point during Extended Limited Transmit signals that a series of duplicate ACKs was caused by reordering and not congestion. Therefore, the receipt of an acceptable ACK that does not carry any SACK information terminates Extended Limited Transmit (step (T.1)). The slow start threshold is set to the maximum of its current value and the current value of cwnd (step (T.3)). Cwnd itself is set to the current value of FlightSize plus one segment (step (T.4)). As a result, the congestion window is not significantly larger than the current amount of outstanding data, so that a burst of data is effectively prevented. If new data is available for transmission and both the new values of cwnd and rwnd allow to send SMSS bytes of previously unsent data, a segment is send (step (T.5)).

On the other hand, if the received ACK acknowledges new data not only cumulatively but also selectively - the ACK carries new SACK information - Extended Limited Transmit is not terminated but re-entered (step (T.1)). If the Cumulative Acknowledgment field of the received ACK covers more than 'recover', one cwnd worth of data has been transmitted during Extended Limited Transmit without any packet loss. Therefore, FlightSizePrev, the amount of outstanding data saved at the beginning of Extended Limited Transmit (step (I.1)), is considered outdated (step (T.2)). This step ensures that in the event of packet loss, the reduction of the cwnd is based on an up-to-

date value, which reflects the number of bytes outstanding in the network (see Section 7). Finally, regardless of whether or not 'recover' is covered, Extended Limited Transmit is re-entered.

The second case that leads to a termination of Extended Limited Transmit is the receipt of an ACK that signals via the algorithm in [RFC6675] that the oldest outstanding segment is considered lost. If either DupThresh or more duplicate ACKs are received, or the oldest outstanding segment is deemed lost via the function IsLost() of [RFC6675], Extended Limited Transmit is terminated and SACK-based loss recovery is entered [RFC6675]. Once the algorithm in [RFC6675] takes over from Extended Limited Transmit, the DupThresh value MUST be held constant until loss recovery is terminated. The process of loss recovery itself is not changed by TCP-aNCR. The only exception is a slight change to either RFC 6675 [RFC6675] or RFC 6937 [RFC6937], depending on whether the PRR algorithm or the traditional Fast Recovery algorithm is used during loss recovery. This change ensures that the adjustment made by the congestion control - the cwnd reduction - is made with respect to the initial amount of outstanding data while Limited Transmit Extended is executed (step (Ret)). The use of FlightSize at this point would no longer be valid since the amount of outstanding data may double by executing Extended Limited Transmit.

7. Discussion of TCP-aNCR

The specification of TCP-aNCR represents an incremental update of RFC 4653 [RFC4653]. All changes made by TCP-aNCR can be divided into two categories. On one hand, they implement TCP-aNCR's ability to dynamically adapted TCP congestion control and loss recovery [RFC5681] to the currently perceived packet reordering on the network path. These include the use of a variable DupThresh and the use of a relative reordering extent. On the other hand, the changes that basically correct weaknesses of the original TCP-NCR algorithm and which are independent of TCP-aNCR adaptability. These include packet reordering during slow start, the prevention of bursts, and the persistent receipt of SACKs.

7.1. Variable Duplicate Acknowledgment Threshold

The central point of the TCP-aNCR algorithm is the usage of a DupThresh that is adaptable to the perceived packet reordering on the network path. Based on the actual amount of outstanding data, TCP-NCR's DupThresh represents roughly the largest amount of time a Fast Retransmit can safely be delayed before a costly retransmission timeout may be triggered. Therefore, to avoid an RTO, TCP-aNCR's reordering-aware DupThresh is an upper bound of the one calculated in TCP-NCR (steps (I.5) and (E.9)). This decouples the avoidance of

spurious Fast Retransmits from the avoidance of RTOs. It allows TCP-aNCR to react fast and efficiently to packet reordering. The DupThresh always corresponds to the minimum of the largest possible and largest detected reordering. With constant packet reordering in terms of the rate and delay, TCP-aNCR gives a DupThresh based on the relative reordering extent with an optimal delay for every bandwidth-delay-product. If TCP-aNCR should not adaptively adjust the DupThresh to the current perceived packet reordering on the network path (because for example an appropriate detection and quantification algorithm is not implemented), the dynamically adaptation of TCP-aNCR can be disabled, so that TCP-aNCR behaves like TCP-NCR [RFC4653].

7.2. Relative Reordering Extent

Whenever a new reordering event is detected and presented to TCP-aNCR in the form of a relative reordering extend 'ReorExtR', TCP-aNCR saves and uses the new 'ReorExtR' if it is larger than the old one (step (EXT)). The upper bound of 1 assures that no excessively large value is used. A 'ReorExtR' larger than one means that more than FlightSize bytes would have been received out-of-order before the reordered segment is received. The delay caused by the reordering is thus longer than the RTT of the TCP connection. Since the RTT is roughly the time a Fast Retransmit can safely be delayed before the retransmission has to be to avoid an RTO, a maximum 'ReorExtR' of one seems to be a suitable value.

The expiration of the retransmission timer is interpreted by TCP-aNCR as an indication of a change in path characteristics, hence, the saved 'ReorExtR' is assumed to be outdated and will be invalidated (step (RTO)). As a consequence, the relative reordering extent 'ReorExtR' increases monotonically between two successive retransmission timeouts and corresponds to the maximum measured reordering extent since the last RTO. Other approaches would be an exponentially-weighted moving average (EWMA) or a histogram of the last n reordering extents. The main drawback of an EWMA is however that on average half of the detected reordering events would be larger than the saved reordering extend. Thus, only half of the spurious retransmits could be avoided. Applying an histogram could largely avoid the disadvantages of an EWMA, however, it would result in a not acceptable increase in memory usage.

In combination with the invalidation after an RTO, the advantage of using maximum is the low complexity as well as its fast convergence to the actual maximum reordering on the network path. As a result, the negative impact that packet reordering has on TCP's congestion control and loss recovery can be avoided. A disadvantage of using a maximum is that if the delay caused by the reordering decreases over the lifetime of the TCP connection, a Fast Retransmit is

unnecessarily long delayed. Nevertheless, since the negative impact reordering has on TCP's congestion control and loss recovery is more substantial than the disadvantage of a longer delay, a decrease of the ReorExtR between RTOs is considered inappropriate.

7.3. Reordering during Slow Start

The arrival of an acceptable ACK during Extended Limited Transmit signals that previously received duplicate ACKs are the result of packet reordering and not congestion, so that Extended Limited Transmit is completed accordingly. Upon the termination of Extended Limited Transmit, and especially when using the Careful variant, TCP-NCR (as well as TCP-aNCR) may be in a situation where the entire cwnd is not being utilized. Therefore, to mitigate a potential burst of segments, in step (T.2) TCP-NCR sets the slow start threshold to the FlightSize that was saved at the beginning of Extended Limited Transmit [RFC4653]. This step should ensure that TCP-NCR slow starts back to the operating point in use before Extended Limited Transmit.

Unfortunately, the assignment in step (T.2) is only correct if the TCP sender already was in congestion avoidance at the time Extended Limited Transmit was entered. Otherwise, if the TCP sender was instead in slow start, the value of ssthresh is greater than the saved FlightSize so that slow start prematurely concludes. This behavior can leave much of the network resources idle, and a long time may be needed in order to use the full capacity. To mitigate this issue, TCP-aNCR sets the slow start threshold to the maximum of its current value and the current cwnd (step (T.3)). This continues slow start after a reordering event happening during slow start.

7.4. Preventing Bursts

In cases where a new single SACK covers more than one segment - this can happen either due to packet loss or packet reordering on the ACK path - TCP-NCR [RFC4653] sends an undesirable burst of data. TCP-aNCR solves this problem by limiting the burst size - the maximum of data that can be sent in response to a single SACK - to the Initial Window [RFC5681] while executing Extended Limited Transmit (steps (E.2), (E.4), and (E.6)). Since IW represents the amount of data that a TCP sender is able to send into the network safely without knowing its characteristics, it is a reasonable value for the burst size, too. If more than IW bytes were SACKed by a single ACK, the additional amount of data becomes available again by the next received duplicate ACK. Thus, the transmission of new segments is spread over the next received ACKs, so that micro bursts - a characteristic of packet reordering in the reverse path - are largely compensated.

Another situation that causes undesired bursts of segments with TCP-NCR is the receipt of an acceptable ACK during Careful Extended Limited Transmit. If multiple segments from a single window of data are delayed by packet reordering, typically the first acceptable ACK after entering the 'disorder' state acknowledges data not only cumulatively but also selectively. Hence, Extended Limited Transmit is not terminated but re-started. If the segments are delayed by the reordering for almost one RTT, then the amount of outstanding data in the network ('pipe') is approximately half the amount of data saved at the beginning of Extended Limited Transmit (FlightSizePrev). If the sequence numbers of the delayed segments are close to each other in the sequence number space, the acceptable ACK acknowledges only a small amount of data, so that FlightSize is still large. As a result, TCP-NCR sets the cwnd to FlightSizePrev in step (T.1). Since 'pipe' is only half of FlightSizePrev due to Careful Extended Limited Transmit, TCP-NCR sends a burst of almost half a cwnd worth of data in the subsequent step (T.3).

Note: Even in the case the sequence numbers of the delayed segments are not close to each other in the sequence number space and cwnd is set in step (T.1) to FlightSize + SMSS, a burst of data will emerge due to re-entering Extended Limited Transmit, because TCP-NCR sets 'skipped' to zero in step (I.2) and uses FlightSizePrev in step (E.2).

TCP-aNCR prevents such a burst by making a clear differentiation between terminating Extended Limited Transmit and a restarting Extended Limited Transmit (step T.1). Only the first case causes the congestion window to be set to the current FlightSize plus one segment. In the latter case, when re-entering Extended Limited Transmit, the congestion window is not adjusted and the original (T.1) of the TCP-NCR specification is omitted. The transmission of new data is then only performed after re-entering Extended Limited Transmit in step (E.2) of the TCP-aNCR specification, where the actual burst mitigation takes place.

7.5. Persistent receiving of Selective Acknowledgments

In some inconvenient cases it could happen that a TCP sender persistently receives SACK information due to reordering on the network path, e.g., if the segments are often and/or lengthy delayed by the packet reordering. With TCP-NCR, the persistent reception of SACKs causes Extended Limited Transmit to be entered with the first received duplicate ACK but never to be terminated if no packet loss occurs - for every received ACK, TCP-NCR either follows steps (E.1) to (E.6) or steps (T.1) to (T.4). In particular, TCP-NCR executes a) for every acceptable ACK step (T.4) and b) at any time step (I.1)

again. Hence, the amount of outstanding data saved at the beginning of Extended Limited Transmit, `FlightSizePrev`, is never updated.

An emerging problem in this context is that during Extended Limited Transmit TCP-NCR determines the transmission of new segments in step (E.2) solely on the basis of `FlightSizePrev`, so that an interim increase of the `cwnd` is not considered (according to [RFC5681], the congestion window is increased for every received acceptable ACK that advances the cumulative ACK point, no matter if it carries SACK information or not). As a result, TCP-NCR can only very slowly determine the available capacity of the communication path.

TCP-aNCR addresses this problem by limiting the amount of data that is allowed to be sent into the network during Extended Limited Transmit not on the basis of `FlightSizePrev`, but on the size of the congestion window. The equation in step E.3 of the TCP-aNCR specification is therefore equal to the one used in [RFC6675] (except for the 'skipped' variable). If an acceptable ACK is received during the execution of Extended Limited Transmit, re-entering Extended Limited Transmit makes any increase in `cwnd` immediately available. Hence, even in the case when persistently receiving SACKs, the available capacity of the communication path can be determined quickly.

Another problem resulting from persistently receiving SACKs, and which is related to the increase in `cwnd` in response to received acceptable ACKs, is the reduction of `cwnd` due to a packet loss. When a packet is considered lost, the congestion control adjustment is done with respect to the amount of outstanding data at the beginning of Extended Limited Transmit, `FlightSizePrev` (step (Ret)). As in the previous case, an increase in `cwnd` is again not taken into account. A simple solution to the problem would be to perform the window reduction not on the basis of `FlightSizePrev` but analogous to step (E.2) based on the current size of `cwnd`.

A problem with this solution is that `cwnd` can potentially be increased, although the TCP connection is limited by the application and not by `cwnd`. Although [RFC2861] specifies that an increase of `cwnd` is only applicable if `cwnd` is fully utilized, this behavior is not specified by any standards track document. But even this conservative increase behavior is guaranteed to not be conservative enough. If, from a single window of data, both segments are delayed but also lost, `cwnd` would first be increased in response to each received acceptable ACKs, while subsequently reduced due to the lost segments, which would not result in a halving of the `cwnd` any more.

The solution proposed by TCP-aNCR reuses the state variable 'recover' from [RFC6582] and adapts the approach taken by NewReno TCP and SACK

TCP to detect, with help of the state variable, the end of one loss recovery phase properly, allowing to recover multiple losses from a single window of data efficiently. Therefore, by entering the 'disorder' state and the starting Extended Limited Transmit, TCP-aNCR saves the highest sequence number sent so far in 'recover'. If a received acceptable ACK covers more than 'recover', one cwnd's worth of data has been transmitted during Extended Limited Transmit without any packet loss. Hence, FlightSizePrev can be updated by 'pipe_max', which reflects the maximum amount of data that is considered to have been in the network during the last RTT. This update takes an interim increase in cwnd into account, so that in case of packet loss, the reduction in cwnd can be based on the current value of FlightSizePrev.

8. Interoperability Issues

TCP-aNCR requires that both the TCP Selective Acknowledgment Option [RFC2018] as well as a SACK-based loss recovery scheme compatible to one given in [RFC6675] are used by the TCP sender. Hence, compatibility to both specifications is REQUIRED.

8.1. Early Retransmit

The specification of TCP-aNCR in this document and the Early Retransmit algorithm specified in [RFC5827] define orthogonal methods to modify DupThresh. Early Retransmit allows the TCP sender to reduce the number of duplicate ACKs required to trigger a Fast Retransmit below the standard DupThresh of three, if FlightSize is less than 4*SMSS and no new segment can be sent. In contrast, TCP-aNCR allows, starting from the minimum of three duplicate ACKs, to increase the DupThresh beyond the standard of three duplicate ACKs to make TCP more robust to packet reordering, if the amount of outstanding data is sufficient to reach the increased DupThresh to trigger Fast Retransmit and Fast Recovery.

8.2. Congestion Window Validation

The increase of the congestion window during application-limited periods can lead to an invalidation of the congestion window, in that it no longer reflects current information about the state of the network, if the congestion window might never have been fully utilized during the last RTT. According to [RFC2861], the congestion window should, first, only be increased during slow-start or congestion avoidance if the cwnd has been fully utilized by the TCP sender and, second, gradually be reduced during each RTT in which the cwnd was not fully used.

A problem that arises in this context is that during Careful Extended Limited Transmit, `cwnd` is not fully utilized due to the variable 'skipped' (see step (E.3)), so that - strictly following [RFC2861] - the congestion window should not be increased upon the receipt of an acceptable ACK. A trivial solution of this problem is to include the variable 'skipped' in the calculation of [RFC2861] to determine whether the congestion window is fully utilized or not.

8.3. Reactive Response to Packet Reordering

As a proactive scheme with the aim to a priori prevent the negative impact that packet reordering has on TCP, TCP-aNCR can conceptually be combined with any reactive response to packet reordering, which attempts to mitigate the negative effects of reordering a posteriori. This is because the modifications of TCP-aNCR to the standard TCP congestion control and loss recovery [RFC6675] are implemented in the 'disorder' state and are performed by the TCP sender before it enters loss recovery, while reactive responses to packet reordering operate generally after entering loss recovery, by undoing the unnecessarily changes to the congestion control state.

If unnecessary changes to the congestion control state are undone after loss recovery, which is typically the case if a spurious Fast Retransmit is detected based on the DSACK option [RFC3708][RFC4015], since first ACK carrying a DSACK option usually arrives at a TCP sender only after loss recovery has already terminated, it might happen that the restoring of the original value of the congestion window is done at a time at which the TCP sender is already back in again in the 'disorder' state and executing Extended Limited Transmit. While this is basically compatible with the TCP-aNCR specification - the undo simply represents an increase of the congestion window - however, some care must be taken that the combination of the algorithms does not lead to unwanted behavior.

8.4. Buffer Auto-Tuning

Although all modifications of the TCP-aNCR algorithm are implemented in the TCP sender, the receiver also potentially has a part to play. If some segments from a single window of data are delayed by the packet reordering in the network, all segments that are received in out-of-order have to be queued in the receive buffer until the holes in sequence number space have been closed and the data can be delivered to the receiving application. In the worst case, which occurs if the TCP sender uses Aggressive Limited Transmit and the reordering delay is close to the RTT, TCP-aNCR increases the receiver's buffering requirement by up to an extra `cwnd`. Therefore, to maximize the benefits from TCP-aNCR, receivers should advertise a large window - ideally by using buffer auto-tuning algorithms - to

absorb the extra out-of-order data. In the case that the additional buffer requirements are not met, the use of the above algorithm takes into account the reduced advertised window - with a corresponding loss in robustness to packet reordering.

9. Related Work

Over the past few years, several solutions have been proposed to improve the performance of TCP in the face of packet reordering. These schemes generally fall into one of two categories (with some overlap): mechanisms that try to prevent spurious retransmits from happening (proactive schemes) and mechanisms that try to detect spurious retransmits and undo the needless congestion control state changes that have been taken (reactive schemes).

[I-D.blanton-tcp-reordering], [ZKFP03] and [LM05] attempt to prevent packet reordering from triggering spurious retransmits by using various algorithms to approximate the DupThresh required to disambiguate loss and reordering over a given network path at a given time. This basic principle is also used in TCP-aNCR. While [I-D.blanton-tcp-reordering] describes four basic approaches on how to increase the DupThresh and discusses pros and cons of these approaches, presents [ZKFP03] a relatively complex algorithm that saves the reordering extents in a histogram and calculates the DupThresh in a way that a certain percentage of samples is smaller than the DupThresh. [LM05] uses an EWMA for the same purpose. Both algorithms do not prevent all the spurious retransmissions by design.

In contrast to the above mentioned algorithms Linux [Linux] implements a proactive scheme by setting the DupThresh to the highest detected reordering and resets only upon an RTO. To avoid a costly retransmission timeout due to the increased DupThresh Linux implements first an extension of the Limited Transmit algorithm, second limits the DupThresh to an upper bound of 127 duplicate ACKs, and third prematurely enters loss recovery if too few segments are in-flight to reach the DupThresh and no additional segments can send. Especially the last change is commendable since, besides TCP-NCR, none of the described algorithms in this section mention a similar concern.

[BHLLO06] and [BSRV04] presents proactive schemes based on timers by which the DupThresh is ignored altogether. After the timer is expired TCP initialize the loss recovery. In [BSRV04] this timer has a length of one RTT and is started when the first duplicate ACK is received, whereas the approach taken in [BHLLO06] solely relies on timers to detect packet loss without taking into account any other congestion signals such as duplicate ACKs. It assigns each segment

send a timestamp and retransmits the segment if the corresponding timer fires.

TCP-NCR [RFC4653] tries to prevent spurious retransmits similar to [I-D.blanton-tcp-reordering] or [ZKFP03] as it delays a retransmission to disambiguate loss and reordering. However, TCP-NCR takes a simplified approach by simply delay a retransmission by an amount based on the current cwnd (in comparison to standard TCP), while the other schemes use relatively complex algorithms in an attempt to derive a more precise value for DupThresh that depends on the current patterns of packet reordering. Many of the features offered by TCP-NCR have been taken into account while designing TCP-aNCR.

Besides the proactive schemes, several other schemes have been developed to detect and mitigate needless retransmissions after the fact. The Eifel detection algorithm [RFC3522], the detection based on DSACKs [RFC3708], and F-RTO scheme [RFC5682] represent approaches to detect spurious retransmissions, while the Eifel response algorithm [RFC4015], [I-D.blanton-tcp-reordering], and Linux [Linux] present respectively implement algorithms to mitigate the changes these events made to the congestion control state. As discussed in Section 8.3 TCP-aNCR could be used in conjunction with these algorithms, with TCP-aNCR attempting to prevent spurious retransmits and some other scheme kicking in if the prevention failed.

10. IANA Considerations

This memo includes no request to IANA.

11. Security Considerations

By taking dedicated actions so that the perceived packet reordering in the network is either underestimating or overestimating by the use of an relative and absolute reordering, an attacker or misbehaving TCP receiver has in regards to TCP's congestion control two options to bias a TCP-aNCR sender. An underestimation of the present packet reordering in the network occurs, if for example, a misbehaving TCP receiver already acknowledges segments while they are actually still in-flight, causing holes premature are closed in the sequence number space of the SACK scoreboard. With regard to TCP-aNCR the result of an underestimated packet reordering is a too small DupThresh, resulting in a premature loss recovery execution. In context of TCP's congestion control the effects of such attacks are limited since the lower bound of TCP-aNCR's DupThresh is the default value of three duplicate ACKs [RFC5681], so that in worst case TCP-aNCR behaves equal to TCP SACK [RFC6675].

In contrast to an underestimation, an overestimation of the packet reordering in the network occurs, if for example, a misbehaving TCP receiver still further send SACKs for subsequent segments before it sends an acceptable ACK for the actually already received delayed segment, so that the hole in the sequence number space of the SACK scoreboard is later closed. In the context of TCP-aNCR the result of such an overestimation is a too large DupThresh, so that in the case of a packet loss TCP's loss recovery is executed later than necessary. Similar to the previous case, the effects of delayed entry into the loss recovery are limited because on the one hand TCP-NCR's DupThresh is used as an upper bound for TCP-aNCR's variable DupThresh so that the entrance to the loss recovery and the adaptation of the congestion window may be delayed at most one RTT. On the other hand, such a limited delay of the congestion control adjustment has even in the worst case only a limited impact on the performance of TCP connection and has generally been regarded as safe for use on the Internet [BBFS01].

12. Acknowledgments

The authors would like to thank Daniel Slot for his TCP-NCR implementation in Linux. We also thank the flowgrind [Flowgrind] authors and contributors for here performance measurement tool, which give us a powerful tool to analyze TCP's congestion control and loss recovery behavior in detail.

13. References

13.1. Normative References

- [I-D.zimmermann-tcpm-reordering-detection]
Zimmermann, A., Schulte, L., Wolff, C., and A. Hannemann, "Detection and Quantification of Packet Reordering with TCP", draft-zimmermann-tcpm-reordering-detection-01 (work in progress), November 2013.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.
- [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.

- [RFC4653] Bhandarkar, S., Reddy, A., Allman, M., and E. Blanton, "Improving the Robustness of TCP to Non-Congestion Events", RFC 4653, August 2006.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, 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.
- [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.
- [RFC6928] Chu, J., Dukkupati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, April 2013.
- [RFC6937] Mathis, M., Dukkupati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, May 2013.

13.2. Informative References

- [BBFS01] Bansal, D., Balakrishnan, H., Floyd, S., and S. Shenker, "Dynamic Behavior of Slowly Responsive Congestion Control Algorithms", Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM'01) pp. 263-274, September 2001.
- [BHLLO06] Bohacek, S., Hespanha, J., Lee, J., Lim, C., and K. Obraczka, "A New TCP for Persistent Packet Reordering", IEEE/ACM Transactions on Networking vol. 2, no. 14, pp. 369-382, April 2006.
- [BSRV04] Bhandarkar, S., Sadry, N., Reddy, A., and N. Vaidya, "TCP-DCR: A Novel Protocol for Tolerating Wireless Channel Errors", IEEE Transactions on Mobile Computing vol. 4, no. 5., pp. 517-529, September 2005.
- [Flowgrind] "Flowgrind Home Page", <<http://www.flowgrind.net>>.
- [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.

- [LM05] Leung, C. and C. Ma, "Enhancing TCP Performance to Persistent Packet Reordering", KICS Journal of Communications and Networks vol. 7, no. 3, pp. 385-393, September 2005.
- [Linux] "The Linux Project", <<http://www.kernel.org>>.
- [RFC0896] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.
- [RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [RFC2861] Handley, M., Padhye, J., and S. Floyd, "TCP Congestion Window Validation", RFC 2861, June 2000.
- [RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L., and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000.
- [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.
- [RFC3708] Blanton, E. and M. Allman, "Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions", RFC 3708, February 2004.
- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.
- [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.
- [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.
- [ZKFP03] Zhang, M., Karp, B., Floyd, S., and L. Peterson, "RR-TCP: A Reordering-Robust TCP with DSACK", Proceedings of the 11th IEEE International Conference on Network Protocols (ICNP'03) pp. 95-106, November 2003.

Appendix A. Changes from previous versions of the draft

This appendix should be removed by the RFC Editor before publishing this document as an RFC.

A.1. Changes from draft-zimmermann-tcpm-reordering-reaction-01

- o Specify interaction between TCP-aNCR and PRR.
- o Fix typo in DupThresh calculation (steps I.5 and E.9).

A.2. Changes from draft-zimmermann-tcpm-reordering-reaction-00

- o Improved the wording throughout the document.
- o Replaced and updated some references.

Authors' Addresses

Alexander Zimmermann
NetApp, Inc.
Sonnenallee 1
Kirchheim 85551
Germany

Phone: +49 89 900594712
Email: alexander.zimmermann@netapp.com

Lennart Schulte
Aalto University
Otakaari 5 A
Espoo 02150
Finland

Phone: +358 50 4355233
Email: lennart.schulte@aalto.fi

Carsten Wolff
credativ GmbH
Hohenzollernstrasse 133
Moenchengladbach 41061
Germany

Phone: +49 2161 4643 182
Email: carsten.wolff@credativ.de

Arnd Hannemann
credativ GmbH
Hohenzollernstrasse 133
Moenchengladbach 41061
Germany

Phone: +49 2161 4643 134
Email: arnd.hannemann@credativ.de