

Internet Draft
draft-cheng-tcpm-fastopen-02.txt
Intended status: Experimental
Expiration date: June, 2012

Y. Cheng
J. Chu
S. Radhakrishnan
A. Jain
Google, Inc.
December 8, 2011

TCP Fast Open

Status of this Memo

Distribution of this memo is unlimited.

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

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

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

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

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

This Internet-Draft will expire in June, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Abstract

TCP Fast Open (TFO) allows data to be carried in the SYN and SYN-ACK packets and consumed by the receiving end during the initial connection handshake, thus providing a saving of up to one full round trip time (RTT) compared to standard TCP requiring a three-way handshake (3WHS) to complete before data can be exchanged.

Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119]. TFO refers to TCP Fast Open. Client refers to the TCP's active open side and server refers to the TCP's passive open side.

1. Introduction

TCP Fast Open (TFO) enables data to be exchanged safely during TCP connection handshake.

This document describes a design that enables qualified applications to attain a round trip saving while avoiding severe security ramifications. At the core of TFO is a security cookie used by the server side to authenticate a client initiating a TFO connection. The document covers the details of exchanging data during TCP's initial handshake, the protocol for TFO cookies, and potential new security vulnerabilities and their mitigation. It also includes discussions on deployment issues and related proposals. TFO requires extensions to the existing socket API, which will be covered in a separate document.

TFO is motivated by the performance need of today's Web applications. Network latency is determined by the round-trip time (RTT) and the number of round trips required to transfer application data. RTT consists of transmission delay and propagation delay. Network bandwidth has grown substantially over the past two decades, much reducing the transmission delay, while propagation delay is largely constrained by the speed of light and has remained unchanged. Therefore reducing the number of round trips has become the most effective way to improve the latency of Web applications [CDCM11].

Standard TCP only permits data exchange after 3WHS [RFC793], which introduces one RTT delay to the network latency. For short transfers, e.g., web objects, this additional RTT becomes a significant portion of the network latency [THK98]. One widely deployed solution is HTTP persistent connections. However, this solution is limited since hosts and middle boxes terminate idle TCP connections due to resource

constraints. E.g., the Chrome browser keeps TCP connections idle up to 5 minutes but 35% of Chrome HTTP requests are made on new TCP connections. More discussions on HTTP persistent connections are in section 7.1.

2. Data In SYN

[RFC793] (section 3.4) already allows data in SYN packets but forbids the receiver to deliver the data to the application until 3WSH is completed. This is because TCP's initial handshake serves to capture

- Old or duplicate SYNs

- SYNs with spoofed IP addresses

TFO allows data to be delivered to the application before 3WSH is completed, thus opening itself to a possible data integrity problem caused by the dubious SYN packets above.

2.1. TCP Semantics and Duplicate SYNs

A past proposal called T/TCP employs a new TCP "TAO" option and connection count to guard against old or duplicate SYNs [RFC1644]. The solution is complex, involving state tracking on per remote peer basis, and is vulnerable to IP spoofing attack. Moreover, it has been shown that even with all the complexity, T/TCP is still not 100% bullet proof. Old or duplicate SYNs may still slip through and get accepted by a T/TCP server [PHRACK98].

Rather than trying to capture all the dubious SYN packets to make TFO 100% compatible with TCP semantics, we've made a design decision early on to accept old SYN packets with data, i.e., to restrict TFO for a class of applications that are tolerant of duplicate SYN packets with data, e.g., idempotent or query type transactions. We believe this is the right design trade-off balancing complexity with usefulness. There is a large class of applications that can tolerate dubious transaction requests.

For this reason, TFO MUST be disabled by default, and only enabled explicitly by applications on a per service port basis.

2.2. SYNs with spoofed IP addresses

Standard TCP suffers from the SYN flood attack [RFC4987] because bogus SYN packets, i.e., SYN packets with spoofed source IP addresses can easily fill up a listener's small queue, causing a service port to be blocked completely until timeouts. Secondary damage comes from faked SYN requests taking up memory space. This is normally not an issue today with typical servers having plenty of memory.

TFO goes one step further to allow server side TCP to process and send up data to the application layer before 3WSH is completed. This opens up much more serious new vulnerabilities. Applications serving ports that have TFO enabled may waste lots of CPU and memory resources processing the requests and producing the responses. If the response is much larger than the request, the attacker can mount an amplified reflection attack against victims of choice beyond the TFO server itself.

Numerous mitigation techniques against the regular SYN flood attack exist and have been well documented [RFC4987]. Unfortunately none are applicable to TFO. We propose a server supplied cookie to mitigate most of the security risks introduced by TFO. A more thorough discussion on SYN flood attack against TFO is deferred to the "Security Considerations" section.

3. Protocol Overview

The key component of TFO is the Fast Open Cookie (cookie), a message authentication code (MAC) tag generated by the server. The client requests a cookie in one regular TCP connection, then uses it for future TCP connections to exchange data during 3WSH:

Requesting Fast Open Cookie:

1. The client sends a SYN with a Fast Open Cookie Request option.
2. The server generates a cookie and sends it through the Fast Open Cookie option of a SYN-ACK packet.
3. The client caches the cookie for future TCP Fast Open connections (see below).

Performing TCP Fast Open:

1. The client sends a SYN with Fast Open Cookie option and data.
2. The server validates the cookie:
 - a. If the cookie is valid, the server sends a SYN-ACK acknowledging both the SYN and the data. The server then delivers the data to the application.
 - b. Otherwise, the server drops the data and sends a SYN-ACK acknowledging only the SYN sequence number.
3. If the server accepts the data in the SYN packet, it may send the response data before the handshake finishes. The max amount is governed by the TCP's congestion control [RFC5681].
4. The client sends an ACK acknowledging the SYN and the server data. If the client's data is not acknowledged, the client retransmits the data in the ACK packet.
5. The rest of the connection proceeds like a normal TCP connection.

The client can perform many TFO operations once it acquires a cookie until the cookie is expired by the server. Thus TFO is useful for applications that have temporal locality on client and server connections.

Requesting Fast Open Cookie in connection 1:

TCP A (Client)		TCP B (Server)
<u>CLOSED</u>		<u>LISTEN</u>
#1 SYN-SENT	----- <SYN, CookieOpt=NIL> ----->	SYN-RCVD
#2 ESTABLISHED (caches cookie C)	<----- <SYN, ACK, CookieOpt=C> -----	SYN-RCVD

Performing TCP Fast Open in connection 2:

TCP A (Client)		TCP B (Server)
<u>CLOSED</u>		<u>LISTEN</u>
#1 SYN-SENT	----- <SYN=x, CookieOpt=C, DATA_A> ----->	SYN-RCVD
#2 ESTABLISHED	<----- <SYN=y, ACK=x+len(DATA_A)+1> -----	SYN-RCVD
#3 ESTABLISHED	<----- <ACK=x+len(DATA_A)+1, DATA_B>-----	SYN-RCVD
#4 ESTABLISHED	----- <ACK=y+1>----->	ESTABLISHED
#5 ESTABLISHED	--- <ACK=y+len(DATA_B)+1>----->	ESTABLISHED

4. Protocol Details

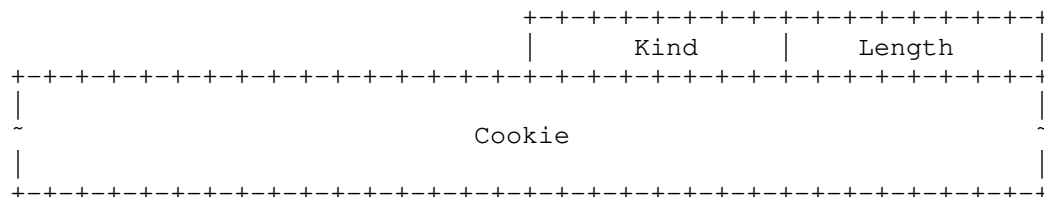
4.1. Fast Open Cookie

The Fast Open Cookie is invented to mitigate new security vulnerabilities in order to enable data exchange during handshake. The cookie is a message authentication code tag generated by the server and is opaque to the client; the client simply caches the cookie and passes it back on subsequent SYN packets to open new connections. The server can expire the cookie at any time to enhance security.

4.1.1. TCP Options

Fast Open Cookie Option

The server uses this option to grant a cookie to the client in the SYN-ACK packet; the client uses it to pass the cookie back to the server in the SYN packet.

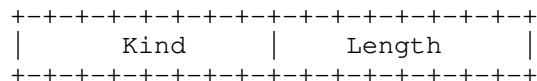


Kind	1 byte: constant TBD (assigned by IANA)
Length	1 byte: range 6 to 18 (bytes); limited by remaining space in the options field. The number MUST be even.
Cookie	4 to 16 bytes (Length - 2)

Options with invalid Length values or without SYN flag set MUST be ignored. The minimum Cookie size is 4 bytes. Although the diagram shows a cookie aligned on 32-bit boundaries, that is not required.

Fast Open Cookie Request Option

The client uses this option in the SYN packet to request a cookie from a TFO-enabled server



Kind 1 byte: same as the Fast Open Cookie option
Length 1 byte: constant 2. This distinguishes the option from
 the Fast Open cookie option.

Options with invalid Length values, without SYN flag set, or with ACK flag set MUST be ignored.

4.1.2. Server Cookie Handling

The server is in charge of cookie generation and authentication. The cookie SHOULD be a message authentication code tag with the following properties:

1. The cookie authenticates the client's (source) IP address of the SYN packet. The IP address can be an IPv4 or IPv6 address.
2. The cookie can only be generated by the server and can not be fabricated by any other parties including the client.
3. The cookie expires after a certain amount of time. The reason is detailed in the "Security Consideration" section. This can be done by either periodically changing the server key used to generate cookies or including a timestamp in the cookie.
4. The generation and verification are fast relative to the rest of SYN and SYN-ACK processing.
5. A server may encode other information in the cookie, and accept more than one valid cookie per client at any given time. But this is all server implementation dependent and transparent to the client.

The server supports the cookie generation and verification operations:

- GetCookie(IP_Address): returns a (new) cookie
- IsCookieValid(IP_Address, Cookie): checks if the cookie is valid, i.e., it has not expired and it authenticates the client IP address.

Example Implementation: a simple implementation is to use AES_128 to encrypt the IPv4 (with padding) or IPv6 address and truncate to 64 bits. The server can periodically update the key to expire the cookies. AES encryption on recent processors is fast and takes only a few hundred nanoseconds [RCCJB11].

Note that if only one valid cookie is allowed per-client and the server can regenerate the cookie independently, the best validation

process may be for the server to simply regenerate a valid cookie and compare it against the incoming cookie. In that case if the incoming cookie fails the check, a valid cookie is readily available to be sent to the client without additional computation.

Also note the server may want to use special cookie values, e.g., "0", for specific scenarios. For example, the server wants to notify the client the support of TFO, but chooses not to return a valid cookie for security or performance reasons upon receiving a TFO request.

4.1.3. Client Cookie Handling

The client **MUST** cache cookies from servers for later Fast Open connections. For a multi-homed client, the cookies are both client and server IP dependent. Beside the cookie, we **RECOMMEND** that the client caches the MSS and RTT to the server to enhance performance.

The MSS advertised by the server is stored in the cache to determine the maximum amount of data that can be supported in the SYN packet. This information is needed because data is sent before the server announces its MSS in the SYN-ACK packet. Without this information, the data size in the SYN packet is limited to the default MSS of 536 bytes [RFC1122]. The client **SHOULD** update the cache MSS value whenever it discovers new MSS value, e.g., through path MTU discovery.

Caching RTT allows seeding a more accurate SYN timeout than the default value [RFC6298]. This lowers the performance penalty if the network or the server drops the SYN packets with data or the cookie options (See "Reliability and Deployment Issues" section below).

The cache replacement algorithm is not specified and is left for the implementations.

Note that before TFO sees wide deployment, clients are advised to also cache negative responses from servers in order to reduce the amount of futile TFO attempts. Since TFO is enabled on a per-service port basis but cookies are independent of service ports, clients' cache should include remote port numbers too.

4.2. Fast Open Protocol

One predominant requirement of TFO is to be fully compatible with existing TCP implementations, both on the client and the server sides.

The server keeps two variables per listening port:

FastOpenEnabled: default is off. It MUST be turned on explicitly by the application. When this flag is off, the server does not perform any TFO related operations and MUST ignore all cookie options.

PendingFastOpenRequests: tracks number of TFO connections in SYN-RCVD state. If this variable goes over a preset system limit, the server SHOULD disable TFO for all new connection requests until PendingFastOpenRequests drops below the system limit. This variable is used for defending some vulnerabilities discussed in the "Security Considerations" section.

The server keeps a FastOpened flag per TCB to mark if a connection has successfully performed a TFO.

4.2.1. Fast Open Cookie Request

Any client attempting TFO MUST first request a cookie from the server with the following steps:

1. The client sends a SYN packet with a Fast Open Cookie Request option.
2. The server SHOULD respond with a SYN-ACK based on the procedures in the "Server Cookie Handling" section. This SYN-ACK SHOULD contain a Fast Open Cookie option if the server currently supports TFO for this listener port.
3. If the SYN-ACK contains a Fast Open Cookie option, the client replaces the cookie and other information as described in the "Client Cookie Handling" section. Otherwise, if the SYN-ACK is first seen, i.e., not a (spurious) retransmission, the client MAY remove the server information from the cookie cache. If the SYN-ACK is a spurious retransmission without valid Fast Open Cookie Option, the client does nothing to the cookie cache for the reasons below.

The network or servers may drop the SYN or SYN-ACK packets with the new cookie options which causes SYN or SYN-ACK timeouts. We RECOMMEND both the client and the server retransmit SYN and SYN-ACK without the cookie options on timeouts. This ensures the connections of cookie requests will go through and lowers the latency penalties (of dropped SYN/SYN-ACK packets). The obvious downside for maximum compatibility is that any regular SYN drop will fail the cookie (although one can argue the delay in the data transmission till after 3WSH is justified if the SYN drop is due to network congestion).

We also RECOMMEND the client to record servers that failed to respond to cookie requests and only attempt another cookie request after certain period.

4.2.2. TCP Fast Open

Once the client obtains the cookie from the target server, the client can perform subsequent TFO connections until the cookie is expired by the server. The nature of TCP sequencing makes the TFO specific changes relatively small in addition to [RFC793].

Client: Sending SYN

To open a TFO connection, the client **MUST** have obtained the cookie from the server:

1. Send a SYN packet.
 - a. If the SYN packet does not have enough option space for the Fast Open Cookie option, abort TFO and fall back to regular 3WHS.
 - b. Otherwise, include the Fast Open Cookie option with the cookie of the server. Include any data up to the cached server MSS or default 536 bytes.
2. Advance to SYN-SENT state and update SND.NXT to include the data accordingly.
3. If RTT is available from the cache, seed SYN timer according to [RFC6298].

To deal with network or servers dropping SYN packets with payload or unknown options, when the SYN timer fires, the client **SHOULD** retransmit a SYN packet without data and Fast Open Cookie options.

Server: Receiving SYN and responding with SYN-ACK

Upon receiving the SYN packet with Fast Open Cookie option:

1. Initialize and reset a local FastOpened flag. If FastOpenEnabled is false, go to step 5.
2. If PendingFastOpenRequests is over the system limit, go to step 5.
3. If IsCookieValid() in section 4.1.2 returns false, go to step 5.
4. Buffer the data and notify the application. Set FastOpened flag and increment PendingFastOpenRequests.
5. Send the SYN-ACK packet. The packet **MAY** include a Fast Open Option. If FastOpened flag is set, the packet acknowledges the SYN and data sequence. Otherwise it acknowledges only the SYN sequence.

The server MAY include data in the SYN-ACK packet if the response data is readily available. Some application may favor delaying the SYN-ACK, allowing the application to process the request in order to produce a response, but this is left to the implementation.

6. Advance to the SYN-RCVD state. If the FastOpened flag is set, the server MAY send more data packets before the handshake completes. The maximum amount is ruled by the initial congestion window and the receiver window [RFC3390].

If the SYN-ACK timer fires, the server SHOULD retransmit a SYN-ACK segment with neither data nor Fast Open Cookie options for compatibility reasons.

Client: Receiving SYN-ACK

The client SHOULD perform the following steps upon receiving the SYN-ACK:

1. Update the cookie cache if the SYN-ACK has a Fast Open Cookie Option.
2. Send an ACK packet. Set acknowledgment number to RCV.NXT and include the data after SND.UNA if data is available.
3. Advance to the ESTABLISHED state.

Note there is no latency penalty if the server does not acknowledge the data in the original SYN packet. The client can retransmit it in the first ACK packet in step 2. The data exchange will start after the handshake like a regular TCP connection.

Server: Receiving ACK

Upon receiving an ACK acknowledging the SYN sequence, the server decrements PendingFastOpenRequests and advances to the ESTABLISHED state. No special handling is required further.

5. Reliability and Deployment Issues

Network or Hosts Dropping SYN packets with data or unknown options

A study [MAF04] found that some middle-boxes and end-hosts may drop packets with unknown TCP options incorrectly. Another study [LANGLEY06] found that 6% of the probed paths on the Internet drop SYN packets with data. The TFO protocol deals with this problem by retransmitting SYN without data or cookie options and we recommend tracking these servers in the client.

Server Farms

A common server-farm setup is to have many physical hosts behind a load-balancer sharing the same server IP. The load-balancer forwards new TCP connections to different physical hosts based on certain load-balancing algorithms. For TFO to work, the physical hosts need to share the same key and update the key at about the same time.

Network Address Translation (NAT)

The hosts behind NAT sharing same IP address will get the same cookie to the same server. This will not prevent TFO from working. But on some carrier-grade NAT configurations where every new TCP connection from the same physical host uses a different public IP address, TFO does not provide latency benefit. However, there is no performance penalty either as described in Section "Client: Receiving SYN-ACK".

6. Security Considerations

The Fast Open cookie stops an attacker from trivially flooding spoofed SYN packets with data to burn server resources or to mount an amplified reflection attack on random hosts. The server can defend against spoofed SYN floods with invalid cookies using existing techniques [RFC4987].

However, the attacker may still obtain cookies from some compromised hosts, then flood spoofed SYN with data and "valid" cookies (from these hosts or other vantage points). With DHCP, it's possible to obtain cookies of past IP addresses without compromising any host. Below we identify two new vulnerabilities of TFO and describe the countermeasures.

6.1. Server Resource Exhaustion Attack by SYN Flood with Valid Cookies

Like regular TCP handshakes, TFO is vulnerable to such an attack. But the potential damage can be much more severe. Besides causing temporary disruption to service ports under attack, it may exhaust server CPU and memory resources.

For this reason it is crucial for the TFO server to limit the maximum number of total pending TFO connection requests, i.e., PendingFastOpenRequests. When the limit is exceeded, the server temporarily disables TFO entirely as described in "Server Cookie Handling". Then subsequent TFO requests will be downgraded to regular connection requests, i.e., with the data dropped and only SYN acknowledged. This allows regular SYN flood defense techniques [RFC4987] like SYN-cookies to kick in and prevent further service disruption.

There are other subtle but important differences in the vulnerability between TFO and regular TCP handshake. Before the SYN flood attack broke out in the late '90s, typical listener's max qlen was small, enough to sustain the highest expected new connection rate and the average RTT for the SYN-ACK packets to be acknowledged in time. E.g., if a server is designed to handle at most 100 connection requests per second, and the average RTT is 100ms, a max qlen on the order of 10 will be sufficient.

This small max qlen made it very easy for any attacker, even equipped with just a dialup modem to the Internet, to cause major disruptions to a web site by simply throwing a handful of "SYN bombs" at its victim of choice. But for this attack scheme to work, the attacker must pick a non-responsive source IP address to spoof with. Otherwise the SYN-ACK packet will trigger TCP RST from the host whose IP address has been spoofed, causing corresponding connection to be removed from the server's listener queue hence defeating the attack. In other words, the main damage of SYN bombs against the standard TCP stack is not directly from the bombs themselves costing TCP processing overhead or host memory, but rather from the spoofed SYN packets filling up the often small listener's queue.

On the other hand, TFO SYN bombs can cause damage directly if admitted without limit into the stack. The RST packets from the spoofed host will fuel rather than defeat the SYN bombs as compared to the non-TFO case, because the attacker can flood more SYNs with data to cost more data processing resources. For this reason, a TFO server needs to monitor the connections in SYN-RCVD being reset in addition to imposing a reasonable max qlen. Implementations may combine the two, e.g., by continuing to account for those connection requests that have just been reset against the listener's PendingFastOpenRequests until a timeout period has passed.

Limiting the maximum number of pending TFO connection requests does make it easy for an attacker to overflow the queue, causing TFO to be disabled. We argue that causing TFO to be disabled is unlikely to be of interest to attackers because the service will remain intact without TFO hence there is hardly any real damage.

6.2. Amplified Reflection Attack to Random Host

Limiting PendingFastOpenRequests with a system limit can be done without Fast Open Cookies and would protect the server from resource exhaustion. It would also limit how much damage an attacker can cause through an amplified reflection attack from that server. However, it would still be vulnerable to an amplified reflection attack from a large number of servers. An attacker can easily cause damage by tricking many servers to respond with data packets at once to any

spoofed victim IP address of choice.

With the use of Fast Open Cookies, the attacker would first have to steal a valid cookie from its target victim. This likely requires the attacker to compromise the victim host or network first.

The attacker here has little interest in mounting an attack on the victim host that has already been compromised. But she may be motivated to disrupt the victim's network. Since a stolen cookie is only valid for a single server, she has to steal valid cookies from a large number of servers and use them before they expire to cause sufficient damage without triggering the defense in the previous section.

One can argue that if the attacker has compromised the target network or hosts, she could perform a similar but simpler attack by injecting bits directly. The degree of damage will be identical, but TFO-specific attack allows the attacker to remain anonymous and disguises the attack as from other servers.

The best defense is for the server not to respond with data until handshake finishes. In this case the risk of amplification reflection attack is completely eliminated. But the potential latency saving from TFO may diminish if the server application produces responses earlier before the handshake completes.

7. Web Performance

7.1. HTTP persistent connection

TCP connection setup overhead has long been identified as a performance bottleneck for web applications [THK98]. HTTP persistent connection was proposed to mitigate this issue and has been widely deployed. However, [RCCJR11][AERG11] show that the average number of transactions per connection is between 2 and 4, based on large-scale measurements from both servers and clients. In these studies, the servers and clients both kept the idle connections up to several minutes, well into the human think time.

Can the utilization rate increase by keeping connections even longer? Unfortunately, this is problematic due to middle-boxes and rapidly growing mobile end hosts. One major issue is NAT. Studies [HNESSK10][MQXMZ11] show that the majority of home routers and ISPs fail to meet the the 124 minutes idle timeout mandated in [RFC5382]. In [MQXMZ11], 35% of mobile ISPs timeout idle connections within 30 minutes. NAT boxes do not possess a reliable mechanism to notify endhosts when idle connections are removed from local tables, either due to resource constraints such as mapping table size, memory, or

lookup overhead, or due to the limited port number and IP address space. Moreover, unmapped packets received by NAT boxes are often dropped silently. (TCP RST is not required by RFC5382.) The end host attempting to use these broken connections are often forced to wait for a lengthy TCP timeout. Thus the browser risks large performance penalty when keeping idle connections open. To circumvent this problem, some applications send frequent TCP keep-alive probes. However, this technique drains power on mobile devices [MQXMZ11]. In fact, power has become a prominent issue in modern LTE devices that mobile browsers close the HTTP connections within seconds or even immediately [SOUDERS11].

Idle connections also consume more memory resources. Due to the complexity of today's web applications, the application layer often needs orders of magnitude more memory than the TCP connection footprint. As a result, servers need to implement advanced resource management in order to support a large number of idle connections.

7.2 Case Study: Chrome Browser

[RCCJR11] studied Chrome browser performance based on 28 days of global statistics. Chrome browser keeps idle HTTP persistent connections up to 5 to 10 minutes. However the average number of the transactions per connection is only 3.3. Due to the low utilization, TCP 3WHS accounts up to 25% of the HTTP transaction network latency. The authors tested a Linux TFO implementation with TFO enabled Chrome browser on popular web sites in emulated environments such as residential broadband and mobile networks. They showed that TFO improves page load time by 10% to 40%. More detailed on the design tradeoffs and measurement can be found at [RCCJB11].

8. TFO's Applicability

TFO aims at latency conscious applications that are sensitive to TCP's initial connection setup delay. These application protocols often employ short-lived TCP connections, or employ long-lived connections but are more sensitive to the connection setup delay due to, e.g., a more strict connection failover requirement.

Only transaction-type applications where RTT constitutes a significant portion of the total end-to-end latency will likely benefit from TFO. Moreover, the client request must fit in the SYN packet. Otherwise there may not be any saving in the total number of round trips required to complete a transaction.

To the extent possible applications protocols SHOULD employ long-lived connections to best take advantage of TCP's built-in congestion control algorithm, and to reduce the impact from TCP's connection

setup overhead. E.g., for the web applications, P-HTTP will likely help and is much easier to deploy hence should be attempted first. TFO will likely provide further latency reduction on top of P-HTTP. But the additional benefit will depend on how much persistency one can get from HTTP in a given operating environment.

One alternative to short-lived TCP connection might be UDP, which is connectionless hence doesn't inflict any connection setup delay, and is best suited for application protocols that are transactional. Practical deployment issues such as middlebox and/or firewall traversal may severely limit the use of UDP based application protocols though.

Note that when the application employs too many short-lived connections, it may negatively impact network stability, as these connections often exit before TCP's congestion control algorithm kicks in. Implementations supporting large number of short-lived connections should employ temporal sharing of TCB data as described in [RFC2140].

More discussion on TCP Fast Open and its projected performance benefit can be found in [RCCJB11].

9. Related Work

9.1. T/TCP

TCP Extensions for Transactions [RFC1644] attempted to bypass the three-way handshake, among other things, hence shared the same goal but also the same set of issues as TFO. It focused most of its effort battling old or duplicate SYNs, but paid no attention to security vulnerabilities it introduced when bypassing 3WHS. Its TAO option and connection count, besides adding complexity, require the server to keep state per remote host, while still leaving it wide open for attacks. It is trivial for an attacker to fake a CC value that will pass the TAO test. Unfortunately, in the end its scheme is still not 100% bullet proof as pointed out by [PHRACK98].

As stated earlier, we take a practical approach to focus TFO on the security aspect, while allowing old, duplicate SYN packets with data after recognizing that 100% TCP semantics is likely infeasible. We believe this approach strikes the right tradeoff, and makes TFO much simpler and more appealing to TCP implementers and users.

9.2. Common Defenses Against SYN Flood Attacks

TFO is still vulnerable to SYN flood attacks just like normal TCP

handshakes, but the damage may be much worse, thus deserves a careful thought.

There have been plenty of studies on how to mitigate attacks from regular SYN flood, i.e., SYN without data [RFC4987]. But from the stateless SYN-cookies to the stateful SYN Cache, none can preserve data sent with SYN safely while still providing an effective defense.

The best defense may be to simply disable TFO when a host is suspected to be under a SYN flood attack, e.g., the SYN backlog is filled. Once TFO is disabled, normal SYN flood defenses can be applied. The "Security Consideration" section contains a thorough discussion on this topic.

9.3. TCP Cookie Transaction (TCPCT)

TCPCT [RFC6013] eliminates server state during initial handshake and defends spoofing DoS attacks. Like TFO, TCPCT allows SYN and SYN-ACK packets to carry data. However, TCPCT and TFO are designed for different goals and they are not compatible.

The TCPCT server does not keep any connection state during the handshake, therefore the server application needs to consume the data in SYN and (immediately) produce the data in SYN-ACK before sending SYN-ACK. Otherwise the application's response has to wait until handshake completes. In contrary, TFO allows server to respond data during handshake. Therefore for many request-response style applications, TCPCT may not achieve same latency benefit as TFO.

Rapid-Restart [SIMPSON11] is based on TCPCT and shares similar goal as TFO. In Rapid-Restart, both the server and the client retain the TCP control blocks after a connection is terminated in order to allow/resume data exchange in next connection handshake. In contrary, TFO does not require keeping both TCB on both sides and is more scalable.

10. IANA Considerations

The Fast Open Cookie Option and Fast Open Cookie Request Option define no new namespace. The options require IANA allocate one value from the TCP option Kind namespace.

11. Acknowledgements

The authors would like to thank Tom Herbert, Rick Jones, Adam Langley, Mathew Mathis, Roberto Peon, and Barath Raghavan for their insightful comments.

12. References

12.1. Normative References

- [RFC793] Postel, J. "Transmission Control Protocol", RFC 793, September 1981.
- [RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [RFC5382] S. Guha, Ed., Biswas, K., Ford B., Sivakumar S., Srisuresh, P., "NAT Behavioral Requirements for TCP", RFC 5382
- [RFC5681] Allman, M., Paxson, V. and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC6298] Paxson, V., Allman, M., Chu, J. and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

12.2. Informative References

- [AERG11] M. Al-Fares, K. Elmeleegy, B. Reed, and I. Gashinsky, "Overclocking the Yahoo! CDN for Faster Web Page Loads". In Proceedings of Internet Measurement Conference, November 2011.
- [CDCM11] Chu, J., Dukkupati, N., Cheng, Y. and M. Mathis, "Increasing TCP's Initial Window", Internet-Draft draft-ietf-tcpm-initcwnd-02.txt (work in progress), October 2011.
- [HNESSK10] S. Haetonen, A. Nyrhinen, L. Eggert, S. Strowes, P. Sarolahti, M. Kojo., "An Experimental Study of Home Gateway Characteristics". In Proceedings of Internet Measurement Conference. October 2010
- [LANGLEY06] Langley, A, "Probing the viability of TCP extensions",
URL <http://www.imperialviolet.org/binary/ecntest.pdf>
- [MAF04] Medina, A., Allman, M., and S. Floyd, "Measuring Interactions Between Transport Protocols and Middleboxes", In Proceedings of Internet Measurement Conference, October 2004.
- [MQXMZ11] Z. Mao, Z. Qian, Q. Xu, Z. Mao, M. Zhang. "An Untold Story of Middleboxes in Cellular Networks", In Proceedings of SIGCOMM. August 2011.

- [PHRACK98] "T/TCP vulnerabilities", Phrack Magazine, Volume 8, Issue 53 artical 6. July 8, 1998. URL <http://www.phrack.com/issues.html?issue=53&id=6>
- [QWGMSS11] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, O. Spatscheck. "Profiling Resource Usage for Mobile Applications: A Cross-layer Approach", In Proceedings of International Conference on Mobile Systems. April 2011.
- [RCCJB11] Radhakrishnan, S., Cheng, Y., Chu, J., Jain, A. and B. Raghavan, "TCP Fast Open". In Proceedings of 7th ACM CoNEXT Conference, December 2011.
- [RFC1644] Braden, R., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, July 1994.
- [RFC2140] Touch, J., "TCP Control Block Interdependence", RFC2140, April 1997.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.
- [RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC6013, January 2011.
- [SIMPSON11] Simpson, W., "Tcp cookie transactions (tcpct) rapid restart", Internet draft draft-simpson-tcpct-rr-02.txt (work in progress), July 2011.
- [SOUDERS11] S. Souders. "Making A Mobile Connection". <http://www.stevesouders.com/blog/2011/09/21/making-a-mobile-connection/>
- [THK98] Touch, J., Heidemann, J., Obraczka, K., "Analysis of HTTP Performance", USC/ISI Research Report 98-463. December 1998.

Author's Addresses

Yuchung Cheng
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
EMail: ycheng@google.com

Jerry Chu
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
EMail: hkchu@google.com

Sivasankar Radhakrishnan
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
EMail: sivasankar@cs.ucsd.edu

Arvind Jain
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043, USA
EMail: arvind@google.com

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

TCP Maintenance and Minor Extensions
(tcpm)
Internet-Draft
Intended status: Experimental
Expires: April 25, 2013

P. Hurtig
Karlstad University
A. Petlund
Simula Research Laboratory AS
M. Welzl
University of Oslo
October 22, 2012

TCP and SCTP RTO Restart
draft-hurtig-tcpm-rtorestart-03

Abstract

This document describes a modified algorithm for managing the TCP and SCTP retransmission timers that provides faster loss recovery when a connection's amount of outstanding data is small. The modification allows the transport to restart its retransmission timer more aggressively in situations where fast retransmit cannot be used. This enables faster loss detection and recovery for connections that are short-lived or application-limited.

Status of this Memo

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

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

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

This Internet-Draft will expire on April 25, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

TCP uses two mechanisms to detect segment loss. First, if a segment is not acknowledged within a certain amount of time, a retransmission timeout (RTO) occurs, and the segment is retransmitted [RFC6298]. While the RTO is based on measured round-trip times (RTTs) between the sender and receiver, it also has a conservative lower bound of 1 second to ensure that delayed segments are not mistaken as lost. Second, when a sender receives duplicate acknowledgments, the fast retransmit algorithm infers segment loss and triggers a retransmission. Duplicate acknowledgments are generated by a receiver when out-of-order segments arrive. As both segment loss and segment reordering cause out-of-order arrival, fast retransmit waits for three duplicate acknowledgments before considering the segment as lost. In some situations, however, the number of outstanding segments is not enough to trigger three duplicate acknowledgments, and the sender must rely on lengthy RTOs for loss recovery.

The amount of outstanding segments can be small for several reasons:

- (1) The connection is limited by the congestion control when the path has a low total capacity (bandwidth-delay product) or the connection's share of the capacity is small. It is also limited by the congestion control in the first RTTs of a connection or after an RTO when the available capacity is probed using slow-start.
- (2) The connection is limited by the receiver's available buffer space.
- (3) The connection is limited by the application if the available capacity of the path is not fully utilized (e.g. interactive applications), or at the end of a transfer, which is frequent if the total amount of data is small (e.g. web traffic).

The first two situations can occur for any flow, as external factors at the network and/or host level cause them. The third situation primarily affects flows that are short or have a low transmission rate. Typical examples of applications that produce short flows are web servers. [RJ10] shows that 70% of all web objects, found at the top 500 sites, are too small for fast retransmit to work. [BPS98]

shows that about 56% of all retransmissions sent by a busy web server are sent after RTO expiry. While the experiments were not conducted using SACK [RFC2018], only 4% of the RTO-based retransmissions could have been avoided. Applications have a low transmission rate when data is sent in response to actions, or as a reaction to real life events. Typical examples of such applications are stock trading systems, remote computer operations and online games. What is special about this class of applications is that they are time-dependant, and extra latency can reduce the application service level [P09]. Although such applications may represent a small amount of data sent on the network, a considerable number of flows have such properties and the importance of low latency is high.

The RTO restart approach outlined in this document makes the RTO slightly more aggressive when the number of outstanding segments is small, in an attempt to enable faster loss recovery for all segments while being robust to reordering. While it still conforms to the requirement in [RFC6298] that segments must not be retransmitted earlier than RTO seconds after their original transmission, it could increase the chance for a spurious timeout, which could degrade performance when the congestion window (cwnd) is large -- for example, when an application sends enough data to reach a cwnd covering 100 segments and then stops. The likelihood and potential impact of this problem as well as possible mitigation strategies are currently under investigation.

While this document focuses on TCP, the described changes are also valid for the Stream Control Transmission Protocol (SCTP) [RFC4960] which has similar loss recovery and congestion control algorithms.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. RTO Restart Overview

The RTO management algorithm described in [RFC6298] recommends that the retransmission timer is restarted when an acknowledgment (ACK) that acknowledges new data is received and there is still outstanding data. The restart is conducted to guarantee that unacknowledged segments will be retransmitted after approximately RTO seconds. However, by restarting the timer on each incoming acknowledgment, retransmissions are not typically triggered RTO seconds after their previous transmission but rather RTO seconds after the last ACK arrived. The duration of this extra delay depends on several factors

but is in most cases approximately one RTT. Hence, in most situations the time before a retransmission is triggered is equal to "RTO + RTT".

The extra delay can be significant, especially for applications that use a lower RT_{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. The RTO restart is performed approximately one RTT after the transmission of the third segment. Thus, if the third segment is lost, as indicated in Figure 1, the effective loss detection time is "RTO + RTT" seconds. In some situations, the effective loss detection time becomes even longer. Consider a scenario where only two segments are outstanding. If the second segment is lost, the time to expire the delayed ACK timer will also be included in the effective loss detection time.

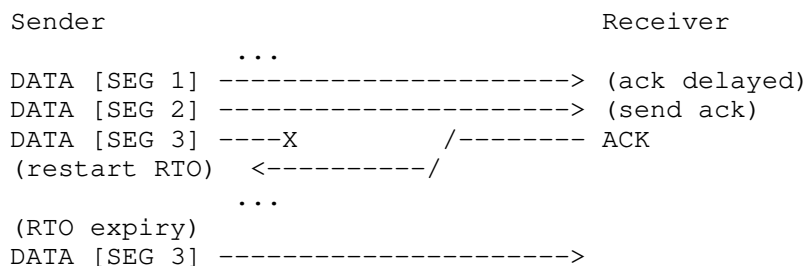


Figure 1: RTO restart example

During normal TCP bulk transfer the current RTO restart approach is not a problem. Actually, as long as enough segments arrive at a receiver to enable fast retransmit, RTO-based loss recovery should be avoided. RTOs should only be used as a last resort, as they drastically lower the congestion window compared to fast retransmit, and the current approach can therefore be beneficial -- it is described in [EL04] to act as a "safety margin" that compensates for some of the problems that the authors have identified with the standard RTO calculation. Notably, the authors of [EL04] also state that "this safety margin does not exist for highly interactive applications where often only a single packet is in flight."

There are only a few situations where timeouts are appropriate, or the only choice. For example, if the network is severely congested and no segments arrive, RTO-based recovery should be used. In this

situation, the time to recover from the loss(es) will not be the performance bottleneck. Furthermore, for connections that do not utilize enough capacity to enable fast retransmit, RTO is the only choice. The time needed for loss detection in such scenarios can become a serious performance bottleneck.

3. RTO Restart Algorithm

To enable faster loss recovery for connections that are unable to use fast retransmit, an alternative RTO restart can be used. By resetting the timer to "RTO - T_earliest", where T_earliest is the time elapsed since the earliest outstanding segment was transmitted, retransmissions will always occur after exactly RTO seconds. This approach makes the RTO more aggressive than the standardized approach in [RFC6298] but still conforms to the requirement in [RFC6298] that segments must not be retransmitted earlier than RTO seconds after their original transmission.

This document specifies the following update of step 5.3 in Section 5 of [RFC6298] (and a similar update in Section 6.3.2 of [RFC4960] for SCTP):

When an ACK is received that acknowledges new data:

- (1) Set T_earliest = 0.
- (2) If the following two conditions hold:
 - (a) The number of outstanding segments is less than four.
 - (b) There is no unsent data ready for transmission or the receiver's advertised window does not permit transmission.set T_earliest to the time elapsed since the earliest outstanding segment was sent.
- (3) Restart the retransmission timer so that it will expire after "RTO - T_earliest" seconds (for the current value of RTO).

The update requires TCP implementations to track the time elapsed since the transmission of the earliest outstanding segment (T_earliest). As the alternative restart is used only when the number of outstanding segments is less than four only four segments need to be tracked. Furthermore, some implementations of TCP (e.g. Linux TCP) already track the transmission times of all segments.

4. Discussion

The currently standardized algorithm has been shown to add at least one RTT to the loss recovery process in TCP [LS00] and SCTP [HB08][PBP09]. Applications that have strict timing requirements (e.g. telephony signaling and gaming) rather than throughput requirements may want to use a lower RTT_{min} than the standard of 1 second [RFC4166]. For such applications the modified restart approach could be important as the RTT and also the delayed ACK timer of receivers will be large components of the effective loss recovery time. Measurements in [HB08] have shown that the total transfer time of a lost segment (including the original transmission time and the loss recovery time) can be reduced with up to 35% using the suggested approach. These results match those presented in [PGH06][PBP09], where the modified restart approach is shown to significantly reduce retransmission latency.

There are several proposals that address the problem of not having enough ACKs for loss recovery. In what follows, we explain why the mechanism described here is complementary to these approaches:

The limited transmit mechanism [RFC3042] allows a TCP sender to transmit a previously unsent segment for each of the first two duplicate acknowledgments. By transmitting new segments, the sender attempts to generate additional duplicate acknowledgments to enable fast retransmit. However, limited transmit does not help if no previously unsent data is ready for transmission or if the receiver is out of buffer space. [RFC5827] specifies an early retransmit algorithm to enable fast loss recovery in such situations. By dynamically lowering the amount of duplicate acknowledgments needed for fast retransmit (dupthresh), based on the number of outstanding segments, a smaller number of duplicate acknowledgments are needed to trigger a retransmission. In some situations, however, the algorithm is of no use or might not work properly. First, if a single segment is outstanding, and lost, it is impossible to use early retransmit. Second, if ACKs are lost, the early retransmit cannot help. Third, if the network path reorders segments, the algorithm might cause more unnecessary retransmissions than fast retransmit.

TCP-NCR [RFC4653] sets the dupthresh to three or more, to better disambiguate reordered and lost segments. In addition, early retransmit lowers the dupthresh when the amount of outstanding data is small, to enable faster loss recovery. The reasons why the RTO restart procedure described in this document does not take dynamic dupthresh considerations into account are twofold. First, if a larger dupthresh is used, the RTO restart approach could be used when the congestion window, and the amount of outstanding data, is larger. However, in such situations the actual amount of outstanding data can

significantly impact the RTT of the connection, making it potentially dangerous to be more aggressive. Second, if a smaller dupthresh is used, the amount of outstanding data needed for a restart is smaller. However, as the congestion window is already small, it does not matter if a retransmission is due to a fast retransmit or an RTO. The resulting congestion window will still be very small, and the only difference is how quickly TCP infers segment loss.

Tail Loss Probe [TLP] is a proposal to send up to two "probe segments" when a timer fires which is set to a value smaller than the RTO. A "probe segment" is a new segment if new data is available, else a retransmission. The intention is to compensate for sluggish RTO behavior in situations where the RTO greatly exceeds the RTT, which, according to measurements reported in [TLP], is not uncommon. The Probe timeout (PTO) is at least 2 RTTs, and only scheduled in case the RTO is farther than the PTO. A spurious PTO is less risky than a spurious RTO, as it would not have the same negative effects (clearing the scoreboard and restarting with slow-start). In contrast, RTO restart is trying to make the RTO more appropriate in cases where there is no need to be overly cautious.

TLP could kick in in situations where RTO restart does not apply, and it could overrule (yielding a similar general behavior, but with a lower timeout) RTO restart in cases where the number of outstanding segments is smaller than 4 and no new segments are available for transmission. The shorter RTO from RTO restart also reduces the probability that TLP is activated because PTO might be farther than RTO. This could make RTO restart more aggressive than the algorithm in [TLP] when:

- (1) no data has been sent in an interval exceeding the RTO
- (2) the number of outstanding segments is 3
- (3) (defined in [RFC5681]) is at least 3

because, under these conditions, in accordance with [RFC5681], 3 packets can immediately be retransmitted, whereas TLP only allows up to two consecutive PTOs.

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

This document discusses a change in how to set the retransmission timer's value when restarted. This change does not raise any new security issues with TCP or SCTP.

7. References

7.1. Normative References

- [RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.
- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.
- [RFC4166] Coene, L. and J. Pastor-Balbas, "Telephony Signalling Transport over Stream Control Transmission Protocol (SCTP) Applicability Statement", RFC 4166, February 2006.
- [RFC4653] Bhandarkar, S., Reddy, A., Allman, M., and E. Blanton, "Improving the Robustness of TCP to Non-Congestion Events", RFC 4653, August 2006.
- [RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC5827] Allman, M., Avrachenkov, K., Ayesta, U., Blanton, J., and P. Hurtig, "Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP)", RFC 5827, May 2010.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

7.2. Informative References

- [BPS98] Balakrishnan, H., Padmanabhan, V., Seshan, S., Stemm, M., and R. Katz, "TCP Behavior of a Busy Web Server: Analysis and Improvements", Proc. IEEE INFOCOM Conf., March 1998.
- [EL04] Ekstroem, H. and R. Ludwig, "The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport", IEEE INFOCOM 2004, March 2004.
- [HB08] Hurtig, P. and A. Brunstrom, "SCTP: designed for timely message delivery?", Springer Telecommunication Systems, May 2010.
- [LS00] Ludwig, R. and K. Sklower, "The Eifel retransmission timer", ACM SIGCOMM Comput. Commun. Rev., 30(3), July 2000.
- [P09] Petlund, A., "Improving latency for interactive, thin-stream applications over reliable transport", Unipub PhD Thesis, Oct 2009.
- [PBP09] Petlund, A., Beskow, P., Pedersen, J., Paaby, E., Griwodz, C., and P. Halvorsen, "Improving SCTP Retransmission Delays for Time-Dependent Thin Streams", Springer Multimedia Tools and Applications, 45(1-3), 2009.
- [PGH06] Pedersen, J., Griwodz, C., and P. Halvorsen, "Considerations of SCTP Retransmission Delays for Thin Streams", IEEE LCN 2006, November 2006.
- [RJ10] Ramachandran, S., "Web metrics: Size and number of resources", Google <http://code.google.com/speed/articles/web-metrics.html>, May 2010.
- [TLP] Dukkipati, N., Cardwell, N., Cheng, Y., and M. Mathis, "TCP Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses", draft-dukkipati-tcpm-tcp-loss-probe-00.txt (work in progress), July 2012.

Authors' Addresses

Per Hurtig
Karlstad University
Universitetsgatan 2
Karlstad, 651 88
Sweden

Phone: +46 54 700 23 35
Email: per.hurtig@kau.se

Andreas Petlund
Simula Research Laboratory AS
P.O. Box 134
Lysaker, 1325
Norway

Phone: +47 67 82 82 00
Email: apetlund@simula.no

Michael Welzl
University of Oslo
PO Box 1080 Blindern
Oslo, N-0316
Norway

Phone: +47 22 85 24 20
Email: michawe@ifi.uio.no

TCP Maintenance Working Group
Internet-Draft
Intended status: Experimental
Expires: August 10, 2013

M. Mathis
N. Dukkipati
Y. Cheng
Google, Inc
Feb 6, 2013

Proportional Rate Reduction for TCP
draft-ietf-tcpm-proportional-rate-reduction-04.txt

Abstract

This document describes an experimental algorithm, Proportional Rate Reduction (PPR) to improve the accuracy of the amount of data sent by TCP during loss recovery. Standard Congestion Control requires that TCP and other protocols reduce their congestion window in response to losses. This window reduction naturally occurs in the same round trip as the data retransmissions to repair the losses, and is implemented by choosing not to transmit any data in response to some ACKs arriving from the receiver. Two widely deployed algorithms are used to implement this window reduction: Fast Recovery and Rate Halving. Both algorithms are needlessly fragile under a number of conditions, particularly when there is a burst of losses such that the number of ACKs returning to the sender is small. Proportional Rate Reduction minimizes these excess window adjustments such that at the end of recovery the actual window size will be as close as possible to ssthresh, the window size determined by the congestion control algorithm. It is patterned after Rate Halving, but using the fraction that is appropriate for target window chosen by the congestion control algorithm.

Status of this Memo

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

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

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

This Internet-Draft will expire on August 10, 2013.

Copyright Notice

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

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

Table of Contents

1. Introduction	3
2. Definitions	5
3. Algorithms	6
3.1. Examples	6
4. Properties	9
5. Measurements	11
6. Conclusion and Recommendations	12
7. Acknowledgements	13
8. Security Considerations	13
9. IANA Considerations	14
10. References	14
10.1. Normative References	14
10.2. Informative References	14
Appendix A. Strong Packet Conservation Bound	15
Authors' Addresses	16

1. Introduction

This document describes an experimental algorithm, Proportional Rate Reduction (PPR) to improve the accuracy of the amount of data sent by TCP during loss recovery.

Standard Congestion Control [RFC5681] requires that TCP (and other protocols) reduce their congestion window in response to losses. Fast Recovery, described in the same document, is the reference algorithm for making this adjustment. Its stated goal is to recover TCP's self clock by relying on returning ACKs during recovery to clock more data into the network. Fast Recovery typically adjusts the window by waiting for one half RTT of ACKs to pass before sending any data. It is fragile because it can not compensate for the implicit window reduction caused by the losses themselves.

RFC 6675 [RFC6675] makes Fast Recovery with SACK [RFC2018] more accurate by computing "pipe", a sender side estimate of the number of bytes still outstanding in the network. With RFC 6675, Fast Recovery is implemented by sending data as necessary on each ACK to prevent pipe from falling below ssthresh, the window size as determined by the congestion control algorithm. This protects Fast Recovery from timeouts in many cases where there are heavy losses, although not if the entire second half of the window of data or ACKs are lost. However, a single ACK carrying a SACK option that implies a large quantity of missing data can cause a step discontinuity in the pipe estimator, which can cause Fast Retransmit to send a burst of data.

The rate-halving algorithm sends data on alternate ACKs during recovery, such that after one RTT the window has been halved. Rate-halving is implemented in Linux after only being informally published [RHweb], including an uncompleted Internet-Draft [RHID]. Rate-halving also does not adequately compensate for the implicit window reduction caused by the losses and assumes a net 50% window reduction, which was completely standard at the time it was written, but not appropriate for modern congestion control algorithms such as Cubic [CUBIC], which reduce the window by less than 50%. As a consequence rate-halving often allows the window to fall further than necessary, reducing performance and increasing the risk of timeouts if there are additional losses.

Proportional Rate Reduction (PPR) avoids these excess window adjustments such that at the end of recovery the actual window size will be as close as possible to ssthresh, the window size determined by the congestion control algorithm. It is patterned after Rate Halving, but using the fraction that is appropriate for the target window chosen by the congestion control algorithm. During PRR one of two additional reduction bound algorithms limits the total window

reduction due to all mechanisms, including transient application stalls and the losses themselves.

We describe two slightly different reduction bound algorithms: conservative reduction bound (CRB), which is strictly packet conserving; and a slow start reduction bound (SSRB), which is more aggressive than CRB by at most one segment per ACK. PRR-CRB meets the Strong Packet Conservation Bound described in Appendix A, however in real networks it does not perform as well as the algorithms described in RFC 6675, which prove to be more aggressive in a significant number of cases. SSRB offers a compromise by allowing TCP to send one additional segment per ACK relative to CRB in some situations. Although SSRB is less aggressive than RFC 6675 (transmitting fewer segments or taking more time to transmit them) it outperforms it, due to the lower probability of additional losses during recovery.

The Strong Packet Conservation Bound on which PRR and both reduction bounds are based is patterned after Van Jacobson's packet conservation principle: segments delivered to the receiver are used as the clock to trigger sending the same number of segments back into the network. As much as possible Proportional Rate Reduction and the reduction bound algorithms rely on this self clock process, and are only slightly affected by the accuracy of other estimators, such as pipe [RFC6675] and cwnd. This is what gives the algorithms their precision in the presence of events that cause uncertainty in other estimators.

The original definition of the packet conservation principle [Jacobson88] treated packets that are presumed to be lost (e.g. marked as candidates for retransmission) as having left the network. This idea is reflected in the pipe estimator defined in RFC 6675 and used here, but it is distinct from Strong Packet Conservation Bound described in Appendix A, which is defined solely on the basis of data arriving at the receiver.

We evaluated these and other algorithms in a large scale measurement study presented in a companion paper [IMC11] and summarized in Section 5. This measurement study was based on RFC 3517 [RFC3517], which has since been superseded by RFC 6675. Since there are slight difference between the two specifications, and we were meticulous about our implementation of RFC 3517 we are not comfortable unconditionally asserting that our measurement results apply to RFC 6675, although we believe this to be the case. We have instead chosen to be pedantic about describing measurement results relative to RFC 3517, on which they were actually based. General discussions algorithms and their properties have been updated to refer to RFC 6675.

We found that for authentic network traffic PRR+SSRB outperforms both RFC 3517 and Linux Rate Halving even though it is less aggressive than RFC 3517. We believe that these results apply to RFC 6675 as well.

The algorithms are described as modifications to RFC 5681 [RFC5681], TCP Congestion Control, using concepts drawn from the pipe algorithm [RFC6675]. They are most accurate and more easily implemented with SACK [RFC2018], but do not require SACK.

2. Definitions

The following terms, parameters and state variables are used as they are defined in earlier documents:

RFC 793: `snd.una`

RFC 5681: duplicate ACK, FlightSize, Sender Maximum Segment Size (SMSS)

RFC 6675: covered (as in "covered sequence numbers")

Voluntary window reductions: choosing not to send data in response to some ACKs, for the purpose of reducing the sending window size and data rate.

We define some additional variables:

SACKd: The total number of bytes that the scoreboard indicates have been delivered to the receiver. This can be computed by scanning the scoreboard and counting the total number of bytes covered by all sack blocks. If SACK is not in use, SACKd is not defined.

DeliveredData: The total number of bytes that the current ACK indicates have been delivered to the receiver. When not in recovery, DeliveredData is the change in `snd.una`. With SACK, DeliveredData can be computed precisely as the change in `snd.una` plus the (signed) change in SACKd. In recovery without SACK, DeliveredData is estimated to be 1 SMSS on duplicate acknowledgements, and on a subsequent partial or full ACK, DeliveredData is estimated to be the change in `snd.una`, minus one SMSS for each preceding duplicate ACK.

Note that DeliveredData is robust: for TCP using SACK, DeliveredData can be precisely computed anywhere in the network just by inspecting the returning ACKs. The consequence of missing ACKs is that later ACKs will show a larger DeliveredData. Furthermore, for any TCP (with or without SACK) the sum of DeliveredData must agree with the

forward progress over the same time interval.

We introduce a local variable "sndcnt", which indicates exactly how many bytes should be sent in response to each ACK. Note that the decision of which data to send (e.g. retransmit missing data or send more new data) is out of scope for this document.

3. Algorithms

At the beginning of recovery initialize PRR state. This assumes a modern congestion control algorithm, CongCtrlAlg(), that might set ssthresh to something other than FlightSize/2:

```
ssthresh = CongCtrlAlg() // Target cwnd after recovery
prrr_delivered = 0       // Total bytes delivered during recovery
prrr_out = 0             // Total bytes sent during recovery
RecoverFS = snd.nxt-snd.una // FlightSize at the start of recovery
```

On every ACK during recovery compute:

```
DeliveredData = change_in(snd.una) + change_in(SACKd)
prrr_delivered += DeliveredData
pipe = (RFC 6675 pipe algorithm)
if (pipe > ssthresh) {
    // Proportional Rate Reduction
    sndcnt = CEIL(prrr_delivered * ssthresh / RecoverFS) - prrr_out
} else {
    // Two version of the reduction bound
    if (conservative) { // PRR+CRB
        limit = prrr_delivered - prrr_out
    } else { // PRR+SSRB
        limit = MAX(prrr_delivered - prrr_out, DeliveredData) + MSS
    }
    // Attempt to catch up, as permitted by limit
    sndcnt = MIN(ssthresh - pipe, limit)
}
```

On any data transmission or retransmission:

```
prrr_out += (data sent) // strictly less than or equal to sndcnt
```

3.1. Examples

We illustrate these algorithms by showing their different behaviors for two scenarios: TCP experiencing either a single loss or a burst of 15 consecutive losses. In all cases we assume bulk data (no application pauses), standard AIMD congestion control and cwnd =

FlightSize = pipe = 20 segments, so ssthresh will be set to 10 at the beginning of recovery. We also assume standard Fast Retransmit and Limited Transmit [RFC3042], so TCP will send two new segments followed by one retransmit in response to the first 3 duplicate ACKs following the losses.

Each of the diagrams below shows the per ACK response to the first round trip for the various recovery algorithms when the zeroth segment is lost. The top line indicates the transmitted segment number triggering the ACKs, with an X for the lost segment. "cwnd" and "pipe" indicate the values of these algorithms after processing each returning ACK. "Sent" indicates how much 'N'ew or 'R'etransmitted data would be sent. Note that the algorithms for deciding which data to send are out of scope of this document.

When there is a single loss, PRR with either of the reduction bound algorithms has the same behavior. We show "RB", a flag indicating which reduction bound subexpression ultimately determined the value of sndcnt. When there is minimal losses "limit" (both algorithms) will always be larger than ssthresh - pipe, so the sndcnt will be ssthresh - pipe indicated by "s" in the "RB" row.

RFC 6675

ack#	X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
cwnd:		20	20	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
pipe:		19	19	18	18	17	16	15	14	13	12	11	10	10	10	10	10	10	10	10
sent:		N	N	R										N	N	N	N	N	N	N

Rate Halving (Linux)

ack#	X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
cwnd:		20	20	19	18	18	17	17	16	16	15	15	14	14	13	13	12	12	11	11
pipe:		19	19	18	18	17	17	16	16	15	15	14	14	13	13	12	12	11	11	10
sent:		N	N	R		N		N		N		N		N		N		N		N

PRR

ack#	X	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
pipe:		19	19	18	18	18	17	17	16	16	15	15	14	14	13	13	12	12	11	10
sent:		N	N	R		N		N		N		N		N		N			N	N
RB:																			s	s

Cwnd is not shown because PRR does not use it.

Key for RB

s:	sndcnt = ssthresh - pipe	// from ssthresh
b:	sndcnt = prr_delivered - prr_out + SMSS	// from banked
d:	sndcnt = DeliveredData + SMSS	// from DeliveredData

(Sometimes more than one applies)

Note that all three algorithms send the same total amount of data. RFC 6675 experiences a "half-window of silence", while the Rate Halving and PRR spread the voluntary window reduction across an entire RTT.

Next we consider the same initial conditions when the first 15 packets (0-14) are lost. During the remainder of the lossy RTT, only 5 ACKs are returned to the sender. We examine each of these algorithms in succession.

RFC 6675

ack#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
cwnd:																20	20	11	11	11
pipe:																19	19	4	10	10
sent:																N	N	7R	R	R

Rate Halving (Linux)

ack#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
cwnd:																20	20	5	5	5
pipe:																19	19	4	4	4
sent:																N	N	R	R	R

PRR-CRB

ack#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
pipe:																19	19	4	4	4
sent:																N	N	R	R	R
RB:																		b	b	b

PRR-SSRB

ack#	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	15	16	17	18	19
pipe:																19	19	4	5	6
sent:																N	N	2R	2R	2R
RB:																		bd	d	d

In this specific situation, RFC 6675 is more aggressive, because once fast retransmit is triggered (on the ACK for segment 17) TCP immediately retransmits sufficient data to bring pipe up to cwnd. Our measurement data (see Section 5) indicates that RFC 6675 significantly outperforms Rate Halving, PRR-CRB and some other similarly conservative algorithms that we tested, showing that it is significantly common for the actual losses to exceed the window

reduction determined by the congestion control algorithm.

The Linux implementation of Rate Halving includes an early version of the conservative reduction bound [RHweb]. In this situation the five ACKs trigger exactly one transmission each (2 new data, 3 old data), and cwnd is set to 5. At a window size of 5, it takes three round trips to retransmit all 15 lost segments. Rate Halving does not raise the window at all during recovery, so when recovery finally completes, TCP will slowstart cwnd from 5 up to 10. In this example, TCP operates at half of the window chosen by the congestion control for more than three RTTs, increasing the elapsed time and exposing it to timeouts in the event that there are additional losses.

PRR-CRB implements a conservative reduction bound. Since the total losses bring pipe below ssthresh, data is sent such that the total data transmitted, prr_out, follows the total data delivered to the receiver as reported by returning ACKs. Transmission is controlled by the sending limit, which was set to prr_delivered - prr_out. This is indicated by the RB:b tagging in the figure. In this case PRR-CRB is exposed to exactly the same problems as Rate Halving, the excess window reduction causes it to take excessively long to recover the losses and exposes it to additional timeouts.

PRR-SSRB increases the window by exactly 1 segment per ACK until pipe rises to ssthresh during recovery. This is accomplished by setting limit to one greater than the data reported to have been delivered to the receiver on this ACK, implementing slowstart during recovery, and indicated by RB:d tagging in the figure. Although increasing the window during recovery seems to be ill advised, it is important to remember that this is actually less aggressive than permitted by RFC 5681, which sends the same quantity of additional data as a single burst in response to the ACK that triggered Fast Retransmit

For less extreme events, where the total losses are smaller than the difference between Flight Size and ssthresh, PRR-CRB and PRR-SSRB have identical behaviours.

4. Properties

The following properties are common to both PRR-CRB and PRR-SSRB except as noted:

Proportional Rate Reduction maintains TCPs ACK clocking across most recovery events, including burst losses. RFC 6675 can send large unclocked bursts following burst losses.

Normally Proportional Rate Reduction will spread voluntary window

reductions out evenly across a full RTT. This has the potential to generally reduce the burstiness of Internet traffic, and could be considered to be a type of soft pacing. Hypothetically, any pacing increases the probability that different flows are interleaved, reducing the opportunity for ACK compression and other phenomena that increase traffic burstiness. However these effects have not been quantified.

If there are minimal losses, Proportional Rate Reduction will converge to exactly the target window chosen by the congestion control algorithm. Note that as TCP approaches the end of recovery `pr_r_delivered` will approach `RecoverFS` and `sndcnt` will be computed such that `pr_r_out` approaches `ssthresh`.

Implicit window reductions due to multiple isolated losses during recovery cause later voluntary reductions to be skipped. For small numbers of losses the window size ends at exactly the window chosen by the congestion control algorithm.

For burst losses, earlier voluntary window reductions can be undone by sending extra segments in response to ACKs arriving later during recovery. Note that as long as some voluntary window reductions are not undone, the final value for `pipe` will be the same as `ssthresh`, the target `cwnd` value chosen by the congestion control algorithm.

Proportional Rate Reduction with either reduction bound improves the situation when there are application stalls (e.g. when the sending application does not queue data for transmission quickly enough or the receiver stops advancing `rwnd`). When there is an application stall early during recovery `pr_r_out` will fall behind the sum of the transmissions permitted by `sndcnt`. The missed opportunities to send due to stalls are treated like banked voluntary window reductions: specifically they cause `pr_r_delivered-pr_r_out` to be significantly positive. If the application catches up while TCP is still in recovery, TCP will send a partial window burst to catch up to exactly where it would have been, had the application never stalled. Although this burst might be viewed as being hard on the network, this is exactly what happens every time there is a partial RTT application stall while not in recovery. We have made the partial RTT stall behavior uniform in all states. Changing this behavior is out of scope for this document.

Proportional Rate Reduction with Reduction Bound is less sensitive to errors in the pipe estimator. While in recovery, `pipe` is intrinsically an estimator, using incomplete information to estimate if un-SACKed segments are actually lost or merely out-of-order in the network. Under some conditions `pipe` can have significant errors, for example `pipe` is underestimated when a burst of reordered data is

prematurely assumed to be lost and marked for retransmission. If the transmissions are regulated directly by pipe as they are with RFC 6675, such as step discontinuity in the pipe estimator causes a burst of data, which can not be retracted once the pipe estimator is corrected a few ACKs later. For PRR, pipe merely determines which algorithm, Proportional Rate Reduction or the reduction bound, is used to compute `sndcnt` from `DeliveredData`. While pipe is underestimated the algorithms are different by at most one segment per ACK. Once pipe is updated they converge to the same final window at the end of recovery.

Under all conditions and sequences of events during recovery, PRR-CRB strictly bounds the data transmitted to be equal to or less than the amount of data delivered to the receiver. We claim that this Strong Packet Conservation Bound is the most aggressive algorithm that does not lead to additional forced losses in some environments. It has the property that if there is a standing queue at a bottleneck with no cross traffic, the queue will maintain exactly constant length for the duration of the recovery, except for ± 1 fluctuation due to differences in packet arrival and exit times. See Appendix A for a detailed discussion of this property.

Although the Strong Packet Conserving Bound is very appealing for a number of reasons, our measurements summarized in Section 5 demonstrate that it is less aggressive and does not perform as well as RFC 6675, which permits large bursts of data when there are bursts of losses. PRR-SSRB is a compromise that permits TCP to send one extra segment per ACK as compared to the packet conserving bound. From the perspective of a strict packet conserving bound, PRR-SSRB does indeed open the window during recovery, however it is significantly less aggressive than RFC6675 in the presence of burst losses.

5. Measurements

In a companion IMC11 paper [IMC11] we describe some measurements comparing the various strategies for reducing the window during recovery. The experiments were performed on servers carrying Google production traffic and are briefly summarized here.

The various window reduction algorithms and extensive instrumentation were all implemented in Linux 2.6. We used the uniform set of algorithms present in the base Linux implementation, including CUBIC [CUBIC], limited transmit [RFC3042], threshold transmit from [FACK] (this algorithm was not present in RFC 3517, but a similar algorithm has been added to RFC 6675) and lost retransmission detection algorithms. We confirmed that the behaviors of Rate Halving (the

Linux default), RFC 3517 and PRR were authentic to their respective specifications and that performance and features were comparable to the kernels in production use. All of the different window reduction algorithms were all present in a common kernel and could be selected with a `sysctl`, such that we had an absolutely uniform baseline for comparing them.

Our experiments included an additional algorithm, PRR with an unlimited bound (PRR-UB), which sends `ssthresh`-pipe bursts when pipe falls below `ssthresh`. This behavior parallels RFC 3517.

An important detail of this configuration is that CUBIC only reduces the window by 30%, as opposed to the 50% reduction used by traditional congestion control algorithms. This accentuates the tendency for RFC 3517 and PRR-UB to send a burst at the point when Fast Retransmit gets triggered because pipe is likely to already be below `ssthresh`. Precisely this condition was observed for 32% of the recovery events: pipe fell below `ssthresh` before Fast Retransmit is triggered, thus the various PRR algorithms start in the reduction bound phase, and RFC 3517 sends bursts of segments with the fast retransmit.

In the companion paper we observe that PRR-SSRB spends the least time in recovery of all the algorithms tested, largely because it experiences fewer timeouts once it is already in recovery.

RFC 3517 experiences 29% more detected lost retransmissions and 2.6% more timeouts (presumably due to undetected lost retransmissions) than PRR-SSRB. These results are representative of PRR-UB and other algorithms that send bursts when pipe falls below `ssthresh`.

Rate Halving experiences 5% more timeouts and significantly smaller final `cwnd` values at the end of recovery. The smaller `cwnd` sometimes causes the recovery itself to take extra round trips. These results are representative of PRR-CRB and other algorithms that implement strict packet conservation during recovery.

6. Conclusion and Recommendations

Although the Strong Packet Conserving Bound used in PRR-CRB is very appealing for a number of reasons, our measurements show that it is less aggressive and does not perform as well as RFC 3517, (and by implication RFC 6675), which permit bursts of data when there are bursts of losses. RFC 3517 and RFC 6675 are conservative in the original sense of Van Jacobson's packet conservation principle, which included the assumption that presumed lost segments have indeed left the network. PRR-CRB makes no such assumption, following instead a

Strong Packet Conserving Bound, in which only packets that have actually arrived at the receiver are considered to have left the network. PRR-SSRB is a compromise that permits TCP to send one extra segment per ACK relative to the Strong Packet Conserving Bound, to partially compensate for excess losses.

From the perspective of the Strong Packet Conserving Bound, PRR-SSRB does indeed open the window during recovery, however it is significantly less aggressive than RFC 3517 (and RFC 6675) in the presence of burst losses. Even so, it often outperforms RFC 3517, (and presumably RFC 6675) because it avoids some of the self inflicted losses caused by bursts.

At this time we see no reason not to test and deploy PRR-SSRB on a large scale. Implementers worried about any potential impact of raising the window during recovery may want to optionally support PRR-CRB (which is actually simpler to implement) for comparison studies. Furthermore, there is one minor detail of PRR that can be improved by replacing `pipe` by `total_pipe` as defined by Laminar TCP [Laminar].

One final comment about terminology: we expect that common usage will drop "slow start reduction bound" from the algorithm name. This document needed to be pedantic about having distinct names for proportional rate reduction and every variant of the reduction bound. However, we do not anticipate any future exploration of the alternative reduction bounds.

7. Acknowledgements

This draft is based in part on previous incomplete work by Matt Mathis, Jeff Semke and Jamshid Mahdavi [RHID] and influenced by several discussion with John Heffner.

Monia Ghobadi and Sivasankar Radhakrishnan helped analyze the experiments.

Ilpo Jarvinen reviewed the code.

Mark Allman improved the document through his insightful review.

8. Security Considerations

Proportional Rate Reduction does not change the risk profile for TCP.

Implementers that change PRR from counting bytes to segments have to

be cautious about the effects of ACK splitting attacks [Savage99], where the receiver acknowledges partial segments for the purpose of confusing the sender's congestion accounting.

9. IANA Considerations

This document makes no request of IANA.

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

10. References

10.1. Normative References

- [RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, October 1996.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.
- [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M., and Y. Nishida, "A Conservative Loss Recovery Algorithm Based on Selective Acknowledgment (SACK) for TCP", RFC 6675, August 2012.

10.2. Informative References

- [RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, January 2001.
- [RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC 3517, April 2003.
- [IMC11] Dukkkipati, N., Mathis, M., and Y. Cheng, "Proportional Rate Reduction for TCP", ACM Internet Measurement Conference IMC11, December 2011.
- [FACK] Mathis, M. and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", ACM SIGCOMM SIGCOMM96, August 1996.
- [RHID] Mathis, M., Semke, J., Mahdavi, J., and K. Lahey, "The Rate-Halving Algorithm for TCP Congestion Control",

draft-mathis-tcp-ratehalving (work in progress),
June 1999.

[RHweb] Mathis, M. and J. Mahdavi, "TCP Rate-Halving with Bounding Parameters", Web publication , December 1997.

[CUBIC] Rhee, I. and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant", PFLDnet 2005, Feb 2005.

[Jacobson88]
Jacobson, V., "Congestion Avoidance and Control", SIGCOMM Comput. Commun. Rev. 18(4), Aug 1988.

[Savage99]
Savage, S., Cardwell, N., Wetherall, D., and T. Anderson, "TCP congestion control with a misbehaving receiver", SIGCOMM Comput. Commun. Rev. 29(5), October 1999.

[Laminar] Mathis, M., "Laminar TCP and the case for refactoring TCP congestion control", draft-mathis-tcpm-tcp-laminar-01 (work in progress), July 2012.

Appendix A. Strong Packet Conservation Bound

PRR-CRB is based on a conservative, philosophically pure and aesthetically appealing Strong Packet Conservation Bound, described here. Although inspired by Van Jacobson's packet conservation principle [Jacobson88], it differs in how it treats segments that are missing and presumed lost. Under all conditions and sequences of events during recovery, PRR-CRB strictly bounds the data transmitted to be equal to or less than the amount of data delivered to the receiver. Note that the effects of presumed losses are included in the pipe calculation, but do not affect the outcome of PRR-CRB, once pipe has fallen below ssthresh.

We claim that this Strong Packet Conservation Bound is the most aggressive algorithm that does not lead to additional forced losses in some environments. It has the property that if there is a standing queue at a bottleneck that is carrying no other traffic, the queue will maintain exactly constant length for the entire duration of the recovery, except for ± 1 fluctuation due to differences in packet arrival and exit times. Any less aggressive algorithm will result in a declining queue at the bottleneck. Any more aggressive algorithm will result in an increasing queue or additional losses if it is a full drop tail queue.

We demonstrate this property with a little thought experiment:

Imagine a network path that has insignificant delays in both directions, except for the processing time and queue at a single bottleneck in the forward path. By insignificant delay, we mean when a packet is "served" at the head of the bottleneck queue, the following events happen in much less than one bottleneck packet time: the packet arrives at the receiver; the receiver sends an ACK; which arrives at the sender; the sender processes the ACK and sends some data; the data is queued at the bottleneck.

If `sndcnt` is set to `DeliveredData` and nothing else is inhibiting sending data, then clearly the data arriving at the bottleneck queue will exactly replace the data that was served at the head of the queue, so the queue will have a constant length. If queue is drop tail and full then the queue will stay exactly full. Losses or reordering on the ACK path only cause wider fluctuations in the queue size, but do not raise its peak size, independent of whether the data is in order or out-of-order (including loss recovery from an earlier RTT). Any more aggressive algorithm which sends additional data will overflow the drop tail queue and cause loss. Any less aggressive algorithm will under fill the queue. Therefore setting `sndcnt` to `DeliveredData` is the most aggressive algorithm that does not cause forced losses in this simple network. Relaxing the assumptions (e.g. making delays more authentic and adding more flows, delayed ACKs, etc) are likely to increase the fine grained fluctuations in queue size but do not change its basic behavior.

Note that the congestion control algorithm implements a broader notion of optimal that includes appropriately sharing the network. Typical congestion control algorithms are likely to reduce the data sent relative to the packet conserving bound implemented by PRR bringing TCP's actual window down to `ssthresh`.

Authors' Addresses

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

Email: mattmathis@google.com

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

Email: nanditad@google.com

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

Email: ycheng@google.com

Congestion Exposure (ConEx)
Internet-Draft
Intended status: Experimental
Expires: May 3, 2012

M. Kuehlewind, Ed.
University of Stuttgart
R. Scheffenegger
NetApp, Inc.
October 31, 2011

Accurate ECN Feedback in TCP
draft-kuehlewind-conex-accurate-ecn-01

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 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.

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 3, 2012.

Copyright Notice

Copyright (c) 2011 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 TCP	4
1.2. Design choices	4
1.3. Requirements Language	5
2. Negotiation in TCP handshake	6
3. Accurate Feedback	7
3.1. Coding	7
3.1.1. Requirements	7
3.1.2. One bit feedback flag	9
3.1.2.1. Discussion	10
3.1.3. Three bit field with counter feedback	11
3.1.3.1. Discussion	12
3.1.4. Codepoints with dual counter feedback	13
3.1.4.1. Implementation	14
3.1.4.2. Discussion	15
3.1.5. Short Summary of the Discussions	16
3.2. TCP Sender	17
3.3. TCP Receiver	17
3.4. Advanced Compatibility Mode	17
4. Acknowledgements	18
5. IANA Considerations	18
6. Security Considerations	18
7. References	19
7.1. Normative References	19
7.2. Informative References	19
Appendix A. Pseudo Code for the Codepoint Coding	19
Authors' Addresses	22

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 discusses and (will in a further version specify) a different scheme for the ECN feedback in the TCP header to provide more than one feedback signal per RTT. This modification does not obsolete [RFC3168]. It provides an extension that requires additional negotiation in the TCP handshake by using the TCP nonce sum (NS) bit which is currently not used when SYN is set.

In the current version of this document there are different coding schemes proposed for discussion. All proposed codings aim to scope with the given bit space. All schemes require the use of the NS bit at least in the TCP handshake. Depending of the coding scheme the accurate ECN feedback extension will or will not include the ECN-Nonce integrity mechanism. A later version of this document will choose between the coding options, and remove the rationale for the choice and the specs of those schemes not chosen. If a scheme will be chosen that does not include ECN Nonce, a mechanism that is requiring a more accurate ECN feedback needs to provide an own method to ensure the integrity of the congestion feedback information or has to scope with the uncertainty of this information.

The following scenarios should briefly show where the accurate feedback is needed or provides additional value:

- a. A Standard TCP sender with [RFC5681] congestion control algorithm that supports ConEx:
In this case the congestion control algorithm still ignores multiple marks per RTT, while the ConEx mechanism uses the extra information per RTT to re-echo more precise congestion information.
- b. A sender using DCTCP without ConEx:
The congestion control algorithm uses the extra info per RTT to perform its decrease depending on the number of congestion marks.
- c. A sender using DCTCP congestion control and supports ConEx:
Both the congestion control algorithm and ConEx use the accurate

ECN feedback mechanism.

- d. A standard TCP sender using RFC5681 congestion control algorithm without ConEx:
No accurate feedback is necessary here. The congestion control algorithm still react only on one signal per RTT. But its best to have one generic feedback mechanism, whether you use it or not.

1.1. Overview ECN and ECN Nonce in 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. 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).

In the TCP header two bits in byte 14 are defined for the use of ECN. The TCP mechanism for signaling the reception of a congestion mark uses the ECN-Echo (ECE) flag in the TCP header. To enable the TCP receiver to determine when to stop setting the ECN-Echo flag, the CWR flag is set by the sender upon reception of the feedback signal.

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. This addition defines the last bit of the 13 byte in the TCP header as the Nonce Sum (NS) bit. With ECN-Nonce a nonce sum is maintain that counts the occurrence of ECT(1) packets.

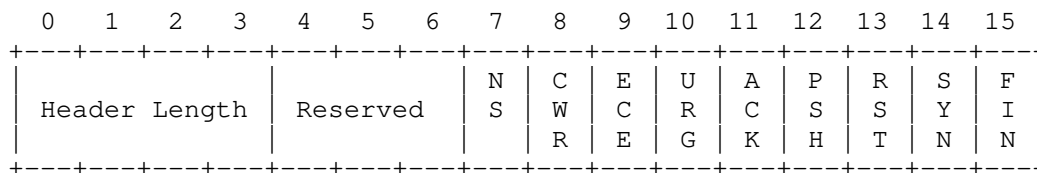


Figure 1: The (post-ECN Nonce) definition of the TCP header flags

1.2. Design choices

The idea of this document is to use the ECE, CWR and NS bits for additional capability negotiation during the SYN/SYN-ACK exchange, and then for the more accurate feedback itself on subsequent packets in the flow (with SYN=0).

Alternatively, a new TCP option could be introduced, to help maintain the accuracy, and integrity of the ECN feedback between receiver and sender. Such an option could provide more information. E.g. ECN for RTP/UDP provides explicit the number of ECT(0), ECT(1), CE, non-ECT marked and lost packets. However, deploying new TCP options has it's own challenges. A seperate documents proposed a new TCP Option for accurate ECN feedback. This option could be used in addition to an more accurate ECN feedback scheme described here or in addition to the classic ECN, when available and needed.

As seen in Figure 1, there are currently three unused flag bits in the TCP header. Any of the below described schemes could be extended by one or more bits, to add higher resiliency against ACK loss. The relative gains would be proportional to each of the described schemes, while the respective drawbacks would remain identical. Thus the approach in this document is to scope with the given number of bits as they seem to be already sufficient and the accurate ECN feedback scheme will only be used instead of the classic ECN and never in parallel.

1.3. 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.

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, we will call the ECN feedback scheme as specified in [RFC3168] the 'classic ECN' and our new proposal the 'accurate ECN feedback' scheme. A 'congestion mark' is defined as an IP packet where the CE codepoint is set. A 'congestion event' refers to one or more congestion marks belong to the same overload situation in the

network (usually during one RTT).

2. Negotiation in TCP handshake

During the TCP hand-shake at the start of a connection, an originator of the connection (host A) MUST indicate a request to get more accurate ECN feedback by setting the TCP flags NS=1, CWR=1 and ECE=1 in the initial SYN.

A responding host (host B) MUST return a SYN ACK with flags CWR=1 and ECE=0. The responding host MUST NOT set this combination of flags unless the preceding SYN has already requested support for accurate ECN feedback as above. Normally a server (B) will reply to a client with NS=0, but if the initial SYN from client A is marked CE, the sever B can set the NS flag to 1 to indicate the congestion immediately instead of delaying the signal to the first acknowledgment when the actually data transmission already started. So, server B MAY set the alternative TCP header flags in its SYN ACK: NS=1, CWR=1 and ECE=0.

The Addition of ECN to TCP SYN/ACK packets is discussed and specified as experimental in [RFC5562]. The addition of ECN to the SYN packet is optional. The security implication when using this option are not further discussed here.

These handshakes are summarized in Table 1 below, with X indicating NS can be either 0 or 1 depending on whether congestion had been experienced. The handshakes used for the other flavors of ECN are also shown for comparison. To compress the width of the table, the headings of the first four columns have been severely abbreviated, as follows:

Ac: *Ac*curate ECN Feedback

N: ECN-*N*once (RFC3540)

E: *E*CN (RFC3168)

I: Not-ECN (*I*mplicit congestion notification).

Ac	N	E	I	[SYN] A->B			[SYN,ACK] B->A			Mode
				NS	CWR	ECE	NS	CWR	ECE	
AB				1	1	1	X	1	0	accurate ECN
A	B			1	1	1	1	0	1	ECN Nonce
A		B		1	1	1	0	0	1	classic ECN
A			B	1	1	1	0	0	0	Not ECN
A			B	1	1	1	1	1	1	Not ECN (broken)

Table 1: ECN capability negotiation between Sender (A) and Receiver (B)

Recall that, if the SYN ACK reflects the same flag settings as the preceding SYN (because there is a broken RFC3168 compliant implementation that behaves this way), RFC3168 specifies that the whole connection MUST revert to Not-ECT.

3. Accurate Feedback

In this section we refer the sender to be the one sending data and the receiver as the one that will acknowledge this data. Of course such a scenario is describing only one half connection of a TCP connection. The proposed scheme, if negotiated, will be used for both half connection as both, sender and receiver, need to be capable to echo and understand the accurate ECN feedback scheme.

3.1. Coding

This section proposes three different coding schemes for discussion. First, requirements are listed that will allow to evaluate the proposed schemes against each other. A later version of this document will choose between the coding options, and remove the rationale for the choice and the specs of those schemes not chosen. The next section provides basically a fourth alternative to allow a compatibility mode when a sender needs accurate feedback but has to operate with a legacy [RFC3168] receiver.

3.1.1. Requirements

The requirements of the accurate ECN feedback protocol for the use of e.g. Conex or DCTCP are to have a fairly accurate (not necessarily perfect), timely and protected signaling. This leads to the following requirements:

Resilience

The ECN feedback signal is implicit carried within the TCP acknowledgment. TCP ACKs can get lost. Moreover, delayed ACK are usually used with TCP. That means in most cases only every second data packets gets acknowledged. In a high congestion situation where most of the packet are marked with CE, an accurate feedback mechanism must still be able to signal sufficient congestion information. Thus the accurate ECN feedback extension has to take delayed ACK and ACK loss into account.

Timely

The CE marking is induced by a network node on the transmission path and echoed by the receiver in the TCP acknowledgment. Thus when this information arrives at the sender, its naturally already about one RTT old. With a sufficient ACK rate a further delay of a small number of ACK can be tolerated but with large delays this information will be out dated due to high dynamic in the network. TCP congestion control which introduces parts of this dynamic operates on an time scale of one RTT. Thus the congestion feedback information should be delivered timely (within one RTT).

Integrity

With ECN Nonce, a misbehaving receiver can be detected with a certain probability. As this accurate ECN feedback might reuse the NS bit it is encouraged to ensure integrity as least as good as ECN Nonce. If this is not possible, alternative approaches should be provided how a mechanism using the accurate ECN feedback extension can re-ensure integrity or give strong incentives for the receiver and network node to cooperate honestly.

Accuracy

Classic ECN feeds back one congestion notification per RTT, as this is supposed to be used for TCP congestion control which reduces the sending rate at most once per RTT. The accurate ECN feedback scheme has to ensure that if a congestion events occurs at least one congestion notification is echoed and received per RRT as classic ECN would do. Of course, the goal of this extension is to reconstruct the number of CE marking more accurately. However, a sender should not assume to get the exact number of congestion marking in a high congestion situation.

Complexity

Of course, the more accurate ECN feedback can also be used, even if only one ECN feedback signal per RTT is need. To enable this proposal for a more accurate ECN feedback as the standard ECN feedback mechanism, the implementation should be as simple as possible and a minimum of addition state information should be needed.

3.1.2. One bit feedback flag

Remark: In one Acknowledgment all acknowledged bytes are regarded as congested

This option is using a one bit flag, namely the ECE bit, to signal more accurate ECN feedback. Other than classic ECN feedback, a accurate ECN feedback receiver MUST set the ECE bit only in one ACK packets for each one CE received. An more accurate ECN feedback receiver MUST NOT wait for a CWR bit from the sender to reset the ECE bit.

As the CWR would now be unused, the CWR MUST be set in the subsequent ACK after the ECE was set.

$$CWR(t) = ECE(t-1)$$

This provides some redundancy in case of ACK loss. If the sender know the ACK'ing scheme of the receiver (e.g. delayed ACKs will send minimum one ACK for every two data packets), the sender can detect ACK loss. If two subsequent ACK or more got lost, the sender SHOULD assume congestion marks for the respective number of ack'ed bytes.

Moreover, when a congestion situation occurs or stops, the receiver MUST immediately acknowledge the data packet and MUST NOT delay the acknowledgment until a further data packet is arrived. A congestion situation occurs when the previous data packet was CE=0 but the current one is CE=1. And a congestion situation stops when the previous data packet was CE=1 and the current one is CE=0.

The following figure shows a simple state machine to describe the receiver behavior.

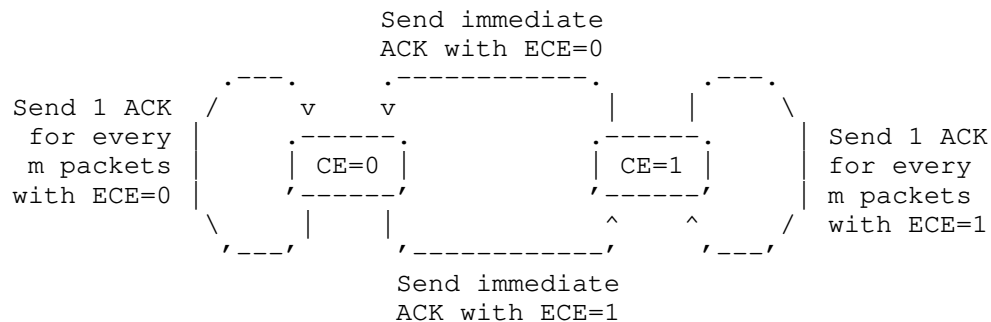


Figure 2: Two state ACK generation state machine

Thus whenever an ACK with the ECE flag set arrives, all acknowledged byte were congestion marked. This scheme provides a byte-wise ECN feedback. The number of CE-marked packet can be estimated by dividing the amount of ack'ed bytes by the Maximum Segment Size (MSS).

When one ACK was lost and the ECN feedback is received based on the CWR set, the sender conservatively SHOULD assume all newly acked bytes as congestion marked.

3.1.2.1. Discussion

ACK loss

In low congestion situations (less than one CE mark per RTT on average), the loss of two subsequent ACKs would result in complete loss of the congestion information. The opposite would be true during high congestion, where the sender can incorrectly assume that all segments were received with the CE codepoint.

One solution would be to carry the same information in a defined number of subsequent ACK packets. This would reduce the number of feedback signals that can be transmitted in one RTT but improve the integrity. More sophisticated solutions based on ACK loss detection might be possible as well.

With DCTCP [Ali10] it was proposed to acknowledge a data packet directly without delay when a congestion situation occurs, as already described above. This scheme allows a more accurate feedback signal in a high congestion/marketing situation. However, using delayed ACKs is important for a variety of reasons, including reducing the load on the data sender.

As this heuristic is triggering immediate ACKs whenever the received

CE bit toggles, arbitrarily large ACK ratios are supported. However, the effective ACK ratio is depending on the congestion state of the network. Thus it may collapse to 1 (one ACK for each data segment). More sophisticated solutions based on ACK loss detection might be possible as well, when every other segment is received with CE set.

ECN Nonce

As the ECN Nonce bit is not used otherwise, ECN Nonce [RFC3540] can be used complementary. Network paths not supporting ECN, misbehaving, or malicious receivers withholding ECN information can therefore be detected.

3.1.3. Three bit field with counter feedback

The receiver maintains an unsigned integer counter which we call ECC (echo congestion counter). This counter maintains a count of how many times a CE marked packet has arrived during the half-connection. Once a TCP connection is established, the three TCP option flags (ECE, CWR and NS) are used as a 3-bit field for the receiver to permanently signal the sender the current value of ECC, modulo 8, whenever it sends a TCP ACK. We will call these three bits the echo congestion increment (ECI) field.

This overloaded use of these 3 option flags as one 3-bit ECI field is shown in Figure 3. The actual definition of the TCP header, including the addition of support for the ECN Nonce, is shown for comparison in Figure 1. This specification does not redefine the names of these three TCP option flags, it merely overloads them with another definition once a flow with accurate ECN feedback is established.

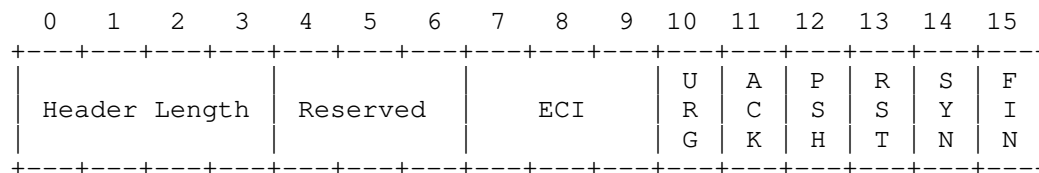


Figure 3: Definition of the ECI field within bytes 13 and 14 of the TCP Header (when SYN=0).

Also note that, whenever the SYN flag of a TCP segment is set (including when the ACK flag is also set), the NS, CWR and ECE flags (i.e. the ECI field of the SYNACK) MUST NOT be interpreted as the 3-bit ECI value, which is only set as a copy of the local ECC value in non-SYN packets.

This scheme was first proposed in [I-D.briscoe-tsvwg-re-ecn-tcp] for the use with re-ECN.

3.1.3.1. Discussion

ACK loss

As pure ACKs are not protected by TCP reliable delivery, we repeat the same ECI value in every ACK until it changes. Even if many ACKs in a row are lost, as soon as one gets through, the ECI field it repeats from previous ACKs that didn't get through will update the sender on how many CE marks arrived since the last ACK got through.

The sender will only lose a record of the arrival of a CE mark if all the ACKs are lost (and all of them were pure ACKs) for a stream of data long enough to contain 8 or more CE marks. So, if the marking fraction was p , at least $8/p$ pure ACKs would have to be lost. For example, if p was 5%, a sequence of 160 pure ACKs (without delayed ACKs) would all have to be lost. When ACK are delay this number has to be reduced by $1/m$. This would still require a sequence of 80 pure lost ACKs with the usual delay rate of $m=2$.

Additionally, to protect against such extremely unlikely events, if a re-ECN sender detects a sequence of pure ACKs has been lost it can assume the ECI field wrapped as many times as possible within the sequence. E.g., if a re-ECN sender receives an ACK with an acknowledgement number that acknowledges L ($>m$) segments since the previous ACK but with a sequence number unchanged from the previously received ACK, it can conservatively assume that the ECI field incremented by $D' = L - ((L-D) \bmod 8)$, where D is the apparent increase in the ECI field. For example if the ACK arriving after 9 pure ACK losses apparently increased ECI by 2, the assumed increment of ECI would still be 2. But if ECI apparently increased by 2 after 11 pure ACK losses, ECI should be assumed to have increased by 10.

ECN Nonce

ECN Nonce cannot be used in parallel to this scheme. But mechanism that make use of this new scheme might provide stronger incentives to declare congestion honestly when needed. E.g. with ConEx each congestion notification suppressed by the receiver should lead the ConEx audit function to discard an equivalent number of bytes such that the receiver does not gain from suppressing feedback. This mechanism would even provide a stronger integrity mechanism than ECN-Nonce does. Without an external framework to discourage the withholding of ECN information, this scheme is vulnerable to the problems described in [RFC3540].

3.1.4. Codepoints with dual counter feedback

In-line with the definition of the previous section in Figure 3, the ECE, CWR and NS bits are used as one field but instead they are encoding 8 codepoints. These 8 codepoints, as shown below, encode either a "congestion indication" (CI) counter or an ECT(1) counter (E1). These counters maintain the number of CE marks or the number of ECT(1) signals observed at the receiver respectively.

ECI	NS	CWR	ECE	CI (base5)	E1 (base3)
0	0	0	0	0	-
1	0	0	1	1	-
2	0	1	0	2	-
3	0	1	1	3	-
4	1	0	0	4	-
5	1	0	1	-	0
6	1	1	0	-	1
7	1	1	1	-	2

Table 2: Codepoint assignment for accurate ECN feedback

By default an accurate ECN receiver MUST echo the CI counter (modulo 5) with the respective codepoints. Whenever an CE occurs and thus the value of the CI has changed, the receiver MUST echo the CI in the next ACK. Moreover, the receiver MUST repeat the codepoint, that provides the CI counter, directly on the subsequent ACK. Thus every value of CI will be transmitted at least twice.

If an ECT(1) mark is receipt and thus E1 increases, the receiver has to convey that updated information to the sender as soon as possible. Thus on the reception of a ECT(1) marked packet, the receiver MUST signal the current value of the E1 counter (modulo 3) in the next ACK, unless a CE mark was receipt which is not echoed yet twice. The receiver MUST also repeat very E1 value. But this repetition does not need to be in the subsequent ACK as the E1 value will only be transmitted when no changes in the CI have occurred. Each E1 value will be send excatly twice. The repetition of every signal will provide further resilience against lost ACKs.

As only a limited number of E1 codepoints exist and the receiver might not acknowledge every single data packet immediately (delayed ACKs), a sender SHOULD NOT mark more than $1/m$ of the packets with ECT(1), where m is the ACK ratio (e.g. 50% when every second data packet triggers an ACK). This constraint will avoid a permanent feedback of E1 only.

This requirement may conflict with delayed ACK ratios larger than two, using the available number of codepoints. A receiver MUST change the ACK'ing rate such that a sufficient rate of feedback signals can be sent. Details on how the change in the ACK'ing rate should be implemented are given in the next subsection.

3.1.4.1. Implementation

The basic idea is for the receiver to count how many packets carry a congestion notification. This could, in principle, be achieved by increasing a "congestion indication" counter (CI.c) for every incoming CE marked segment. Since the space for communicating the information back to the sender in ACKs is limited, instead of directly increasing this counter, a "gauge" (CI.g) is increased instead.

When sending an ACK, the content of this gauge (capped by the maximum number that can be encoded in the ACK, e.g. 4 for CI, and 2 for E1) is copied to the actual counter, and CI.g is reduced by the value that was copied over and transmitted, unless CI.g was zero before. To avoid losing information, it is ensured that an ACK is sent at least after 5 incoming congestion marks (i.e. when CI.g exceeds 5).

For resilience against lost ACKs, an indicator flag (CI.i) ensures that, whether another congestion indication arrives or not, a second ACK transmits the previous counter value again.

The same counter / gauge method is used to count and feed back (using a different mapping) the number of incoming packets marked ECT(1) (called E1 in the algorithm). As fewer codepoints are available for conveying the E1 counter value, an immediate ACK MUST be triggered whenever the gauge E1.g exceeds a threshold of 3. The sender receives the receiver's counter values and compares them with the locally maintained counter. Any increase of these counters is added to the sender's internal counters, yielding a precise number of CE-marked and ECT(1) marked packets. Architecturally the counters never decrease during a TCP session. However, any overflow must be modulo 5 for CI, and modulo 3 for E1.

The following table provides an example showing an half-connection with an TCP sender A and receiver B. The sender maintains a counter CI.r to reconstruct the number of CE mark receipt at receiver-side.

	Data	TCP A	IP	TCP B	Data
		SEQ ACK CTL		SEQ ACK CTL	
--		-----	-----	-----	
1		0100 SYN CWR,ECE,NS	---->		
2			<----	0300 0101 SYN ACK,CWR	
3		0101 0301 ACK	ECT0 -CE->	CI.c=0 CI.g=1	
4	100	0101 0301 ACK	ECT0 ---->	CI.c=1 CI.g=0	
5			<----	0301 0201 ACK ECI=CI.1	
6	100	CI.r=1 0201 0301 ACK	ECT0 -CE->	CI.c=1 CI.g=1	
7	100	0301 0301 ACK	ECT0 -CE->	CI.c=1 CI.g=2	
8			XX--	0301 0401 ACK ECI=CI.1	
9	100	CI.r=1 0401 0301 ACK	ECT0 -CE->	CI.c=1 CI.g=3	
10	100	0501 0301 ACK	ECT0 -CE->	CI.c=5 CI.g=0	
11			<----	0301 0601 ACK ECI=CI.0	
12	100	CI.r=5 0601 0301 ACK	ECT0 -CE->	CI.c=5 CI.g=1	
13	100	0701 0301 ACK	ECT0 -CE->	CI.c=5 CI.g=2	
14			<----	0301 0801 ACK ECI=CI.0	
		CI.r=5			

Table 3: Codepoint signal example

3.1.4.2. Discussion

ACK loss

As this scheme sends each codepoint (of the two subsets) at least two times, at least one, and up to two consecutive ACKs can be lost. Further refinements, such as interleaving ACKs when sending

codepoints belonging to the two subsets (e.g. CI, E1), can allow the loss of any two consecutive ACKs, without the sender losing congestion information, at the cost of also reducing the ACK ratio.

At low congestion rates, the sending of the current value of the CI counter by default allows higher numbers of consecutive ACKs to be lost, without impacting the accuracy of the ECN signal.

ECN Nonce

By comparing the number of incoming ECT(1) notifications with the actual number of packets that were transmitted with an ECT(1) mark as well as the sum of the sender's two internal counters, the sender can probabilistic detect a receiver that would send false marks or suppress accurate ECN feedback, or a path that doesn't properly support ECN.

This approach maintains a balanced selection of properties found in ECN Nonce, Section 3.1.3, and Section 3.1.2. A delayed ACK ratio of two can be sustained indefinitely even during heavy congestion, but not during excessive ECT(1) marking, which is under the control of the sender. An higher ACK ratios can be sustained even when congestion is low but its need for the E1 feedback.

3.1.5. Short Summary of the Discussions

With the exception of the signaling scheme described in Section 3.1.2, all signaling may fail to work, if middleboxes intervene and check on the semantic of [RFC3168] signals.

The scheme described in Section 3.1.4 is the most complex to implement especially on a receiver, with much additional state to be kept there, compared to the other signaling schemes. With the advances in compute power, many more cycles are available to process TCP than ever before.

Table 4 gives an overview of the relative implications of the different proposed signaling schemes. Further discussion should be included here in the next version of this document.

Section	Resiliency	Timely	Integrity	Accuracy	Complexity
1-bit-flag	-	+	+	-	+
3-bit-field	++	++	--	++	-
Codepoints	+	+	+	++	--

Table 4: Overview of accurate feedback schemes

Whereas the first scheme is the simplest one (and also provides byte-wise feedback which might be preferable), it has a drawback with respect to reliability. The second one is the most reliable but does not provide an integrity mechanism.

3.2. TCP Sender

This section will specify the sender-side action describing how to exclude the accurate number of congestion markings from the given receiver feedback signal.

When the accurate ECN feedback scheme is supported by the receiver, the receiver will maintain an echo congestion counter (ECC). The ECC will hold the number of CE marks received. A sender that is understanding the accurate ECN feedback will be able to reconstruct this ECC value on the sender side by maintaining a counter ECC.r.

On the arrival of every ACK, the sender calculates the difference D between the local ECC.r counter, and the signaled value of the receiver side ECC counter. The value of ECC.r is increased by D , and D is assumed to be the number of CE marked packets that arrived at the receiver since it sent the previously received ACK.

3.3. TCP Receiver

This section will describe the receiver-side action to signal the accurate ECN feedback back to the sender. In any case the receiver will need to maintain a counter of how many CE marking has been seen during a connection. Depending on the chosen coding scheme there will be different action to set the corresponding bits in the TCP header. For all case it might be helpful if the receiver is able to switch from a delayed ACK behavior to send ACKs immediately after the data packet reception in a high congestion situation.

3.4. Advanced Compatibility Mode

This section describes a possibility to achieve more accurate feedback even when the receiver is not capable of the new accurate ECN feedback scheme with the drawback of less reliability.

During initial deployment, a large number of receivers will only support [RFC3168] classic ECN feedback. Such a receiver will set the ECE bit whenever it receives a segment with the CE codepoint set, and clear the ECE bit only when it receives a segment with the CWR bit set. As the CE codepoint has priority over the CWR bit (Note: the wording in this regard is ambiguous in [RFC3168], but the reference

implementation of ECN in ns2 is clear), a [RFC3168] compliant receiver will not clear the ECE bit on the reception of a segment, where both CE and CWR are set simultaneously. This property allows the use of a compatibility mode, to extract more accurate feedback from legacy [RFC3168] receivers by setting the CWR permanently.

Assuming an delayed ACK ratio of one, a sender can permanently set the CWR bit in the TCP header, to receive a more accurate feedback of the CE codepoints as seen at the receiver. This feedback signal is however very brittle and any ACK loss may cause congestion information to become lost. Delayed ACKs and ACK loss can both not be accounted for in a reliable way, however. Therefore, a sender would need to use heuristics to determine the current delay ACK ratio m used by the receiver (e.g. most receivers will use $m=2$), and also the recent ACK loss ratio (l). Acknowledge Congestion Control (AckCC) as defined in [RFC5690] can not be used, as deployment of this feature is only experimental.

Using a phase locked loop algorithm, the CWR bit can then be set only on those data segments, that will trigger a (delayed) ACK. Thereby, no congestion information is lost, as long as the ACK carrying the ECE bit is seen by the sender.

Whenever the sender sees an ACK with ECE set, this indicates that at least one, and at most $m / (m - 1)$ data segments with the CE codepoint set where seen by the receiver. The sender SHOULD react, as if m CE indications where reflected back to the sender by the receiver, unless additional heuristics (e.g. dead time correction) can determine a more accurate value of the "true" number of received CE marks.

4. Acknowledgements

We want to thank Michael Welzl and Bob Briscoe for their input and discussion.

5. IANA Considerations

This memo includes no request to IANA.

6. Security Considerations

For coding schemes that increase robustness for the ECN feedback, similar considerations as in RFC3540 apply for the selection of when to sent a ECT(1) codepoint.

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.
- [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.
- [RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, February 2010.

Appendix A. Pseudo Code for the Codepoint Coding

Receiver:

Input signals: CE , ECT(1)

TCP Fields: ECI (3-bit field from CWR and ECE). CI.cm and El.cm map into these 8 codepoints (ie. 5 and 3 codepoints)

These counters get tracked by the following variables:

CI.c (congestion indication - counter, modulo a multiple of the available codepoints to represent CI.c in the ECI field.
 Range[0..n*CI.cp-1])
 CI.g (congestion indication - gauge, [0.."inf"])
 CI.i (congestion indication - iteration, [0,1])
 These are to track CE indications.

El.c, El.g and El.r (doing the same, but for ECT(1) signals).

Constants:

CI.cp (number of codepoints available to signal)
 CI.cm[] (codepoint mapping for CI)
 El.cp (number of codepoints available for El signal)
 El.cm[0..(El.cp-1)] (codepoint mappings for El)

At session initialization, all these counters are set to 0;

When a Segment (Data, ACK) is received,
 perform the following steps:

If a CE codepoint is received,
 Increase CI.g by 1
 If a ECT(1) codepoint is received,
 Increase El.g by 1
 If (CI.g > 5) # When ACK rate is not sufficient to keep
 or (El.g > 3) # gauge close to zero, increase ACK rate
 # works independent of delACK number (ie AckCC)
 Cancel pending delayed ACK (ACK this segment immediately)
 # this increases the ACK rate to a maximum of 1.5 data segments
 # per ACK, with delACK=2,
 # and CE mark rate exceeds 75% for a number
 # of at least 18 segments.
 # 5 codepoints would allow delack=2 indefinitely btw

When preparing an ACK to be sent:

If (CI.g > 0) or
 ((El.i != 0) and (CI.i != 0)) # El.g = 0 is to skip this
 # if only the 2nd CI.c ACK
 # has to be sent - effectively alternating CI.c and El.c on ACKs
 # should give slightly better resiliency against ack losses
 If CI.i == 0 # updates to CI.c allowed
 and CI.g > 0 # update is meaningful
 CI.i = 1 # may be larger
 # if more resiliency is reqd
 CI.c += min(CI.cp-1, CI.g) # CI.cp-1 is 3 for 4 codepoints,

```

                                # 4 for 5 etc
CI.c = CI.c modulo CI.cp*CI.cp # using modulo the square of
                                # available codepoints,
                                # for convinience (debugging)
CI.g -= min(CI.cp-1,CI.g)      #
Else
CI.i--                          # just in case CI.f was set to
                                # more than 1 for resiliency
Send next ACK with ECI = CI.cm[CI.c modulo CI.cp]
Else
If (El.g > 0) or (El.i != 0)

If (El.i == 0) and (El.g > 0)
El.i = 1
El.c += min(El.cp-1,El.g)
El.c = El.c modulo El.cp*El.cp
El.g -= min(El.cp-1,El.g)
Else
El.i--
Send next ACK with ECI = El.cm[El.c modulo El.cp]
Else
Send next ACK with ECI = CI.cm[CI.c modulo CI.cp] # default action

Sender:

Counters:

CI.r - current value of CEs seen by receiver
El.s - sum of all sent ECT(1) marked packets (up to snd.nxt)
El.s(t) - value of El.s at time (in sequence space) t
El.r - value signaled by receiver about received ECT(1) segments
El.r(t) - value of El.r at time (in sequence space) t
CI.r(t) - ditto
```

```
# Note: With a codepoint-implementation,
# a reverse table ECI[n] -> CI.r / El.r is needed.
# This example is simplified with 4/4 codepoints
# instead of 5/3

If ACK with NS=0
CI.r += (ECI + 4 - (CI.r mod CI.cp)) mod CI.cp
# The wire protocol transports the absolute value
# of the receiver-side counter.
# Thus the (positive only) delta needs to be calculated,
# and added to the sender-side counter.
If ACK with NS=1
El.r += (ECI + 4 - (El.r mod El.cp)) mod El.c

# Before CI.r or El.r reach a (binary) rollover,
# they need to roll over some multiple of CI.cp
# and El.cp respectively.

CI.r = CI.r modulo CI.cp * n_CI
El.r = El.r modulo El.cp * n_El

# (an implementation may choose to use a single constant,
# ie 3^4*5^4 for 16-bit integers,
# or 3^8*5^8 for 32-bit integers)

# The following test can (probabilistically) reveal,
# if the receiver or path is not properly
# handling ECN (CE, El) marks

If not El.r(t) <= El.s(t) <= El.r(t) + CI.r(t)
# -> receiver lies (or too many ACKs got lost,
# which can be checked too by the sender).
```

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

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

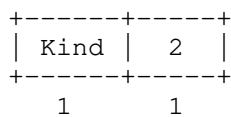


Figure 1: Accurate ECN feedback TCP option negotiation

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 TSecr field in the first <SYN>, which is currently unused. This specification defines the semantics of the TSecr 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

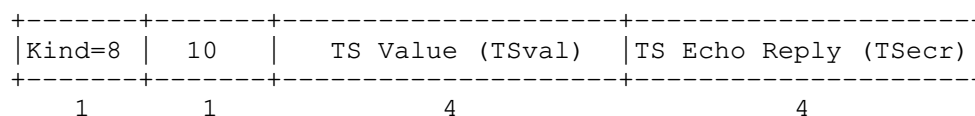


Figure 1: RFC1323 TSopt

"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

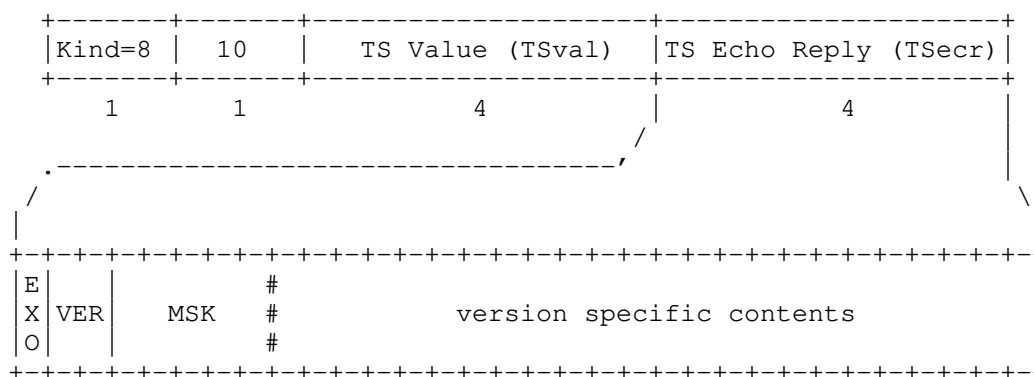


Figure 2: Timestamp Capability flags

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

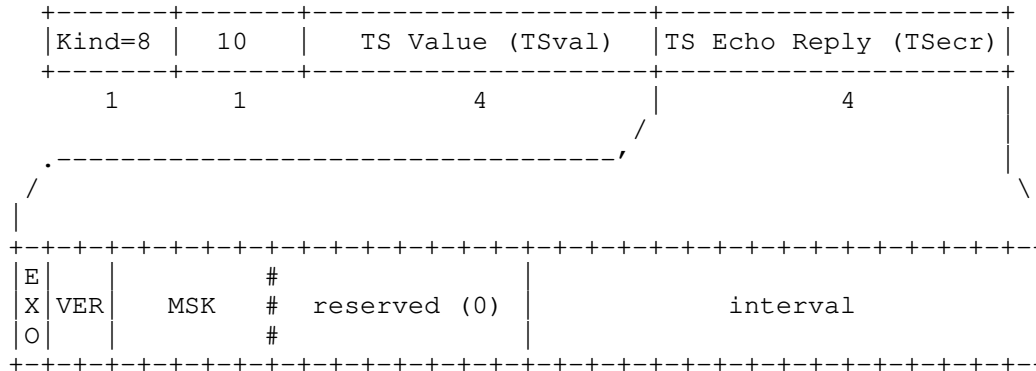


Figure 3: Timestamp Capability flags - version 0

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](http://caia.swin.edu.au/reports/100219A/CAIA-TR-100219A.pdf)>.
- [CUBIC] Rhee, I., Ha, S., and L. Xu, "CUBIC: A New TCP-Friendly
High-Speed TCP Variant", Feb 2005, <[http://
citeseerx.ist.psu.edu/viewdoc/
download?doi=10.1.1.153.3152&rep=rep1&type=pdf](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.153.3152&rep=rep1&type=pdf)>.
- [Cho08] Cho, I., Han, J., and J. Lee, "Enhanced Response Algorithm
for Spurious TCP Timeout (ER-SRTO)", Jan 2008, <<http://>

ubinet.yonsei.ac.kr/v2/publication/hpmn_papaers/ic/
2008_Enhanced%20Response%20Algorithm%20for%20Spurious%
20TCP.pdf>.

[I-D.blanton-tcp-reordering]

Blanton, E., Dimond, R., and M. Allman, "Practices for TCP
Senders in the Face of Segment Reordering",
draft-blanton-tcp-reordering-00 (work in progress),
February 2003.

[I-D.ietf-tcpm-1323bis]

Borman, D., Braden, R., Jacobson, V., and R.
Scheffenegger, "TCP Extensions for High Performance",
draft-ietf-tcpm-1323bis-04 (work in progress),
August 2012.

[I-D.ietf-tcpm-anumita-tcp-stronger-checksum]

Biswas, A., "Support for Stronger Error Detection Codes in
TCP for Jumbo Frames",
draft-ietf-tcpm-anumita-tcp-stronger-checksum-00 (work in
progress), May 2010.

[I-D.ietf-tcpm-tcp-security]

Gont, F., "Survey of Security Hardening Methods for
Transmission Control Protocol (TCP) Implementations",
draft-ietf-tcpm-tcp-security-03 (work in progress),
March 2012.

[I-D.sabatini-tcp-sack]

Sabatini, A., "Highly Efficient Selective Acknowledgement
(SACK) for TCP", draft-sabatini-tcp-sack-01 (work in
progress), August 2012.

[Linux]

Sarolahti, P., "Linux TCP", Apr 2007,
<<http://www.cs.clemson.edu/~westall/853/linuxtcp.pdf>>.

[PH04]

Eckstroem, H. and R. Ludwig, "The Peak-Hopper: A New End-
to-End Retransmission Timer for Reliable Unicast
Transport", Apr 2004, <[citeseerx.ist.psu.edu/viewdoc/
download?doi=10.1.1.76.2748&rep=rep1&type=pdf](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.76.2748&rep=rep1&type=pdf)>.

[Path99]

Allman, M. and V. Paxson, "On Estimating End-to-End
Network Path Properties", Sep 1999,
<<http://www.icir.org/mallman/papers/estimation.ps>>.

[RFC1122]

Braden, R., "Requirements for Internet Hosts -
Communication Layers", STD 3, RFC 1122, October 1989.

- [RFC2883] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, July 2000.
- [RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.
- [RFC3522] Ludwig, R. and M. Meyer, "The Eifel Detection Algorithm for TCP", RFC 3522, April 2003.
- [RFC4015] Ludwig, R. and A. Gurtov, "The Eifel Response Algorithm for TCP", RFC 4015, February 2005.
- [RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, August 2007.
- [RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC 6013, January 2011.
- [RFC6247] Eggert, L., "Moving the Undeployed TCP Extensions RFC 1072, RFC 1106, RFC 1110, RFC 1145, RFC 1146, RFC 1379, RFC 1644, and RFC 1693 to Historic Status", RFC 6247, May 2011.
- [RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

Appendix A. Possible use cases

A.1. Timestamp clock rate exposure

Today, each TCP host may use an arbitrary, locally defined clock source to derive the timestamp value from. Even though only a handful of typically used clock rates are implemented in common TCP stacks, this does not guarantee that any future stack will choose the same clock rate. This poses a problem for current state of the art heuristics, which try to determine the senders timestamp clock rate by pure passive observation of the TCP stream, and affects both advanced heuristics in the partner host of a TCP session, or arbitrarily located passive observation points to estimate TCP session parameters.

The proposed mechanism would reveal this information explicitly, even though other environmental factors, such as the operation of a TCP stack in a virtualized environment, may result in some deviations in the actually used clock rate.

High-speed and real-time stacks would be expected to operate with higher clock rates, while the observed variance in (known) timestamp clock vs. reference clock could help in determining between physical and virtual end hosts, for example.

A.2. Early spurious retransmit detection

Using the provided timestamp negotiation scheme, clients utilizing slow running timestamp clocks can set aside a small number of least significant bits in the timestamps. These bits can be used to differentiate between original and retransmitted segments, even within the same timestamp clock tick (i.e. when RTT is shorter than the TCP timestamp clock interval). It is recommended to use only a single bit (mask = 1), unless the sender can also perform lost retransmission detection. Using more than 2 bits for this purpose is discouraged due to the diminishing probability of losing retransmitted packets more than one time. A simple scheme could send out normal data segments with the so masked bits all cleared. Each advance of the timestamp clock also clears those bits again. When a segment is retransmitted without the timestamp clock increasing, these bits increased by one for each consecutive retry of the same segment, until the maximum value is reached. Newly sent segments (during the same clock interval) should maintain these bits, in order to maintain monotonically increasing values, even though compliant end hosts do not require this property. This scheme maintains monotonically increasing timestamp values - including the masked bits. Even without negotiating the immediate mirroring of timestamps (done by simultaneously doing timestamp capabilities negotiation, and selective acknowledgments), this extends the use of the Eifel Detection [RFC3522] and Eifel Response [RFC4015] algorithm to detect and react to spurious retransmissions under all circumstances. Also, currently experimental schemes such as ER-SRTO [Cho08] could be deployed without requiring the receiver to explicitly support that capability.

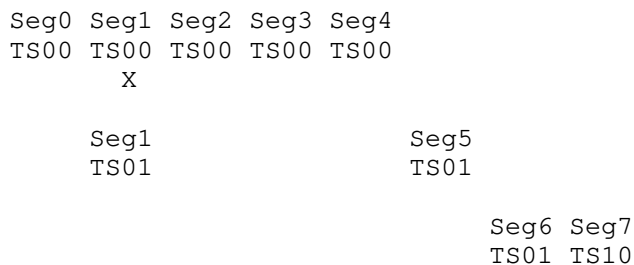


Figure 4: timestamp for spurious retransmit detection

Masked bits are the 2nd digit, the timestamp value is represented by the first digit. The timestamp clock "ticks" between segment 6 and 7.

A.3. Early lost retransmission detection

During phases where multiple segments in short succession (but not necessarily successive segments) are lost, there is a high likelihood that at least one segment is retransmitted, while the cause of loss (i.e. congestion, fading) is still persisting. The best current algorithms can recover such a lost retransmission with a few constraints, for example, that the session has to have at least DupThresh more segments to send beyond the current recovery phase. During loss recovery, when a retransmission is lost again, currently the timestamp can also not be used as means of conveying additional information, to allow more rapid loss recovery while maintaining packet conservation principles. Only the timestamp of the last segment preceding the continuous loss will be reflected. Using the extended timestamp option negotiation together with selective acknowledgements, the receiver will immediately reflect the timestamp of the last seen segment. Using both SACK and TS information in conjunction with each other, a sender can infer the exact order in which original and retransmitted segments are received. This allows faster recovery from lost retransmissions while maintaining the principle of packet conservations and avoiding costly retransmission timeouts.

The implementation can be done in combination with the masked bit approach described in the previous paragraph, or without. However, if the timestamp clock interval is lower than $1/2$ RTT, both the original and the retransmitted segment may carry an identical timestamp. If the sender cannot discriminate between the original and the retransmitted segments, it must refrain from taking any action before such a determination can be made.

In this example, masked bits are used, with a simple marking method. As the timestamp value of the retransmission itself is already different from the original segments, such an additional discrimination would not strictly be required here. The timestamp clock ticks in the first digit and the dupthresh value is 3.

Seg0	Seg1	Seg2	Seg3	Seg4	Seg5	Seg6	Seg7
TS00	TS00	TS00	TS10	TS10	TS10	TS10	TS20
	X	X	X	*			
Seg1	Seg2	Seg3	Seg4				
TS21	TS30	TS30	TS30				
X							
Seg1						Seg8	Seg9
TS31						TS31	TS40

Figure 5: timestamp under loss

If Seg1,TS00 is lost twice, and Seg4,TS10 is also lost, the sender could resend Seg1 once more after observing dupthresh number of segments sent after the first retransmission of Seg1 being received (ie, when Seg4 is SACKed). However, there is an ambiguity between retransmitted segments and original segments, as the sender cannot know, if a SACK for one particular segment was due to the retransmitted segment, or a delayed original segment. The timestamp value will not help in this case, as per RFC1323 it will be held at TS00 for the entire loss recovery episode. Therefore, currently a sender has to assume that any SACKed segments may be due to delayed original sent segments, and can only resolve this conflict by injecting additional, previously unsent segments. Once dupthresh newly injected segments are SACKed, continuous loss (and not further delay) of Seg1 can safely be assumed, and that segment be resent. This approach is conservative but constrained by the requirement that additional segments can be sent, and thereby delayed in the response.

With the simultaneous use of timestamp extended options together with selective acknowledgments, the receiver would immediately reflect back the timestamp of the last received segment. This allows the sender to discriminate between a SACK due to a delayed Seg4,TS10, or a SACK because of Seg4,TS30. Therefore, the appropriate decision (retransmission of Seg1 once more, or addressing the observed reordering/delay accordingly [I-D.blanton-tcp-reordering]) can be taken with high confidence.

A.4. Integrity of the Timestamp value

If the timestamp is used for congestion control purposes, an incentive exists for malicious receivers to reflect tampered timestamps, as demonstrated with some exploits [CUBIC].

One way to address this is to not use timestamp information directly, but to keep state in the sender for each sent segment, and track the round trip time independent of sent timestamps. Such an approach has

the drawback, that it is not straightforward to make it work during loss recovery phases for those segments possibly lost (or reordered). In addition there is processing and memory overhead to maintain possibly extensive lists in the sender that need to be consulted with each ACK. Despite these drawbacks, this approach is currently implemented due to lack of alternatives (see [Linux], and [BSD10]).

The preferred approach is that the sender MAY choose to protect timestamps from such modifications by including a fingerprint (secure hash of some kind) in some of the least significant bits. However, doing so prevents a receiver from using the timestamp for other purposes, unless the receiver has prior knowledge about this use of some bits in the timestamp value. Furthermore, strict monotonic increasing values are still to be maintained. That constraint restricts this approach somewhat and limits or inhibits the use of timestamp values for direct use by the receiver (i.e. for one-way delay variation measurement, as the hash bits would look like random noise in the delay measurement).

A.5. Disambiguation with slow Timestamp clock

In addition, but somewhat orthogonal to maintaining timestamp value integrity, there is a use case when the sender does not support a timestamp clock interval that can guarantee unique timestamps for retransmitted segments. This may happen whenever the TCP timestamp clock interval is higher than the round-trip time of the path. For unambiguously identifying regular from retransmitted segments, the timestamp must be unique for otherwise identical segments. Reserving the least significant bits for this purpose allows senders with slow running timestamp clocks to make use of this feature. However, without modifying the receiver behavior, only limited benefits can be extracted from such an approach. Furthermore the use of this option has implications in the protection against wrapped sequence numbers (PAWS - [RFC1323]), as the more bits are set aside for tamper prevention, the faster the timestamp number space cycles.

Using Timestamp capabilities to explicitly negotiate mask bits, and set aside a (low) number of least significant bits for the above listed purposes, allows a sender to use more reliable integrity checks. These masked bits are not to be considered part of the timestamp value, for the purposes described in [RFC1323] (i.e. PAWS) and subsequent heuristics using timestamp values (i.e. Eifel Detection), thereby lifting the strict requirement of always monotonically increasing timestamp values. However, care should be taken to not mask too many bits, for the reasons outlined in [RFC1323]. Using a mask value higher than 8 is therefore discouraged.

The reason for having 5 bits for the mask field nevertheless is to allow the implementation of this protocol in conjunction with TCP cookie transaction (TCPCT) extended timestamps [RFC6013]. That allows for nearly a quarter of a 128 bit timestamp to be set aside.

A.6. Masked timestamps as segment digest

After making TCP alternate checksums historic (see [RFC6247]), there still remains a need to address increased corruption probabilities when segment sizes are increased (see [I-D.ietf-tcpm-anumita-tcp-stronger-checksum]).

Utilizing a completely masked TSval field allows the sender to include a stronger CRC32, with semantics independent of the fixed TCP header fields. However, such a use would again exclude the use of PAWS on the receiver side, and a receiver would need to know the specifics of the digest for processing. It is assumed, that such a digest would only cover the data payload of a TCP segment. In order to allow disambiguation of retransmissions, a special TSval can be defined (e.g. TSval=0) which bypasses regular CRC processing but allows the identification of retransmitted segments.

The full semantics of such a data-only CRC scheme are beyond the scope of this document, but would require a different version of the timestamp capability. Nevertheless, allowing the full TSval to remain unprocessed by the receiver for the purpose of PAWS even in version 0 could still allow the successful negotiation of sender-side enhancements such as loss recovery improvements (see Appendix A.2, and Appendix A.3).

In effect, the masked portion of the timestamp value represent an unreliable out of band signal channel, that could also be used for other purposes than solely performing timestamp integrity checks (for example, this would allow ER-SRTO algorithms [Cho08]).

Appendix B. Open Issues

- o The split between this draft and [I-D.trammell-tcpm-timestamp-interval] is cursory; additional separation of timestamp interval export may be necessary.
- o [bht] suggest changing the "versioning" construct to a "capabilities" construct, especially since two bits of versioning might as well be none. The base specification would then define the alternate semantics WRT SACK and could use capabilities to define further semantics.
- o [bht] does it make sense to move masking out of the base spec and into the 8 "unused" bits in "version 0" (in order to get more capabilities bits / "magic bits" to reduce erroneous negotiation)?
- o [bht] does it make sense to define SACK-echo as version/capability independent?

Appendix C. Revision history

This appendix should be removed by the RFC Editor before publishing this document as a RFC.

00 ... initial draft, early submission to meet deadline.

01 ... refined draft, focusing only on those capabilities that have an immediate use case. Also excluding flags that can be substituted by other means (MIR - synergistic with SACK option only, RNG moved to appendix A, BIA removed and the exponent bias set to a fixed value. Also extended other paragraphs.

02 ... updated document after IETF80 - referrals to "timestamp options" were seen to be ambiguous with "timestamp option", and therefore replaced by "timestamp capabilities". Also, the document was reworked to better align with RFC4101. Removed SGN and increased FRAC to allow higher precision.

03 ... removed references to "opaque" and "transparent". substituted "timestamp clock interval" for all instances of rate. Changed signal encoding to resemble a scale/value approach like what is done with Window Scaling. As added benefit, clock quality can be implicitly signaled, since multiple representations can map to identical time intervals. Added discussion around resilience against broken RFC1323 implementations (Win95, Linux 2.3.41+), which deviate from expected Timestamp signaling behavior.

04 ... removed previous appendix A (range negotiation); minor edit to improve wording; moved Section 6 to the Appendix, and removed covert channels from the potential uses; added some text to discuss future versioning (compatible and incompatible variants); changed document structure; added guidance around PAWS; added pseudo-code examples (probably to be removed again)

05 ... added new Open Issues section, added reference to separate interval draft, removed content on timestamp interval exposure which now appears in the interval draft. Removed pseudocode examples until they can be reworked on finalization of the mechanism, as they refer to fields which have changed / moved to the interval draft.

Authors' Addresses

Richard Scheffenegger
NetApp, Inc.
Am Euro Platz 2
Vienna, 1120
Austria

Phone: +43 1 3676811 3146
Email: rs@netapp.com

Mirja Kuehlewind
University of Stuttgart
Pfaffenwaldring 47
Stuttgart 70569
Germany

Email: mirja.kuehlewind@ikr.uni-stuttgart.de

Brian Trammell
Swiss Federal Institute of Technology Zurich
Gloriastrasse 35
8092 Zurich
Switzerland

Phone: +41 44 632 70 13
Email: trammell@tik.ee.ethz.ch

TCPM Working Group
Internet Draft
Intended status: Standards Track
Expires: January 2013

J. Touch
USC/ISI
July 16, 2012

Automating the Initial Window in TCP
draft-touch-tcpm-automatic-iw-03.txt

Status of this Memo

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

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

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

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

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

This Internet-Draft will expire on January 16, 2011.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in

Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Abstract

The Initial Window (IW) provides the starting point for TCP's feedback-based congestion control algorithm. Its value has increased over time to increase performance and to reflect increased capability of Internet devices. This document describes a mechanism to adjust the IW over long timescales, to make future changes more safely deployed and to potentially avoid reexamination of this value in the future.

Table of Contents

1. Introduction.....	2
2. Conventions used in this document.....	3
3. Design Considerations.....	3
4. Proposed IW Algorithm.....	4
5. Discussion.....	7
6. Observations.....	8
7. Security Considerations.....	9
8. IANA Considerations.....	9
9. Conclusions.....	9
10. References.....	9
10.1. Normative References.....	9
10.2. Informative References.....	9
11. Acknowledgments.....	10

1. Introduction

TCP's congestion control algorithm uses an initial window value (IW), both as a starting point for new connections and after one RTO or more [RFC2581][RFC2861]. This value has evolved over time, originally one maximum segment size (MSS), and increased to the lesser of four MSS or 4,380 bytes [RFC3390][RFC5681]. For typical Internet connections with an maximum transmission units (MTUs) of 1500 bytes, this permits three segments of 1,460 bytes each.

The IW value was originally implied in the original TCP congestion control description, and documented as a standard in 1997 [RFC2001][Ja88]. The value was last updated in 1998 experimentally, and moved to the standards track in 2002 [RFC2414][RFC3390]. There have been recent proposals to update the IW based on further increases in host and router capabilities and network capacity, some

focusing on specific values (e.g., IW=10), and others prescribing a schedule for increases over time (e.g., IW=6 for 2011, increasing by 1-2 MSS per year).

This document proposes that TCP can objectively measure when an IW is too large, and that such feedback should be used over long timescales to adjust the IW automatically. The result should be safer to deploy and might avoid the need to repeatedly revisit IW size over time.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

3. Design Considerations

TCP's IW value has existed statically for over two decades, so any solution to adjusting the IW dynamically should have similarly stable, non-invasive effects on the performance and complexity of TCP. In order to be fair, the IW should be similar for most machines on the public Internet. Finally, a desirable goal is to develop a self-correcting algorithm, so that IW values that cause network problems can be avoided. To that end, we propose the following list of design goals:

- o Impart little to no impact to TCP in the absence of loss, i.e., it should not increase the complexity of default packet processing in the normal case.
- o Adapt to network feedback over long timescales, avoiding values that persistently cause network problems.
- o Decrease the IW in the presence of sustained loss of IW segments, as determined over a number of different connections.

- o Increase the IW in the absence of sustained loss of IW segments, as determined over a number of different connections.
- o Operate conservatively, i.e., tend towards leaving the IW the same in the absence of sufficient information, and give greater consideration to IW segment loss than IW segment success.

We expect that, without other context, a good IW algorithm will converge to a single value, but this is not required. An endpoint with additional context or information, or deployed in a constrained environment, can always use a different value. In specific, information from previous connections, or sets of connections with a similar path, can already be used as context for such decisions [RFC2140].

However, if a given IW value persistently causes packet loss during the initial burst of packets, it is clearly inappropriate and could be inducing unnecessary loss in other competing connections. This might happen for sites behind very slow boxes with small buffers, which may or may not be the first hop.

4. Proposed IW Algorithm

Below is a simple description of the proposed IW algorithm. It relies on the following parameters:

- o MinIW = 3 MSS or 4,380 bytes (as per RFC3390)
- o MaxIW = 10
- o MulDecr = 0.5
- o AddIncr = 2 MSS
- o Threshold = 0.05

We assume that the minimum IW (MinIW) should be as currently specified [RFC3390]. The maximum IW can be set to a fixed value [Ch10], or set based on a schedule if trusted time references are available [Al10]; here we prefer a fixed value. We also propose to use an AIMD algorithm, with increase and decreases as noted.

Although these parameters are somewhat arbitrary, their initial values are not important except that the algorithm is AIMD and the MaxIW should not exceed that recommended for other systems on the Internet. Current proposals, including default current operation, are degenerate cases of the algorithm below for given parameters -

notably `MulDec = 1.0` and `AddIncr = 0 MSS`, thus disabling the automatic part of the algorithm.

The proposed algorithm is as follows:

0. On boot:

```
IW = MaxIW; # assume this is in bytes, and an even number of MSS
```

1. Upon starting a new connection

```
CWND = IW;
conncount++;
IWnotchecked = 1; # true
```

2. During a connection's SYN-ACK processing, if SYN-ACK includes ECN, treat as if the IW is too large

```
if (IWnotchecked && (synackecn == 1)) {
    losscount++;
    IWnotchecked = 0; # never check again
}
```

3. During a connection, if retransmission occurs, check the seqno of the outgoing packet (in bytes) to see if the resent segment fixes an IW loss:

```
if (Retransmitting && IWnotchecked && ((ISN - seqno) < IW)) {
    losscount++;
    IWnotchecked = 0; # never do this entire "if" again
} else {
    IWnotchecked = 0; # you're beyond the IW so stop checking
}
```

4. Once every 1000 connections, as a separate process (i.e., not as part of processing a given connection):

```
if (conncount > 1000) {  
    if (losscount/conncount > threshold) {  
        # the number of connections with errors is too high  
        IW = IW * MulDecr;  
    } else {  
        IW = IW + AddIncr;  
    }  
}
```

We recognize that this algorithm can yield a false positive when the sequence number wraps around. This can be avoided using either PAWS [RFC1323] context or 64-bit internal sequence numbers (as in TCP-AO [RFC5925]). Alternately, false positives can be allowed since they are expected to be infrequent and thus will not affect the overall statistics of the algorithm.

The following additional constraints are imposed:

>> The automatic IW algorithm MUST initialize to MaxIW, in the absence of other context information.

If there are too few connections to make a decision, or if there is otherwise insufficient information to increase the IW, then the MaxIW defaults to the current recommended value.

>> An implementation may allow the MaxIW to grow beyond the currently recommended Internet default, but not more than 2 segments per calendar year.

If an endpoint has a persistent history of successfully transmitting IW segments without loss, then it is allowed to probe the Internet to determine if larger IW values have similar success. This probing is limited and requires a trusted time source, otherwise the MaxIW remains constant.

>> An implementation MUST adjust the IW based on loss statistics at least once every 1000 connections.

An endpoint needs to be sufficiently reactive to IW loss.

>> An implementation MUST decrease the IW by at least one MSS when indicated during an evaluation interval.

An endpoint that detects loss needs to decrease its IW by at least one MSS, otherwise it is not participating in an automatic reactive algorithm.

>> An implementation MUST increase by no more than 2 MSS per evaluation interval.

An endpoint that does not experience IW loss needs to probe the network incrementally.

>> An implementation SHOULD use an IW that is an integer multiple of 2 MSS.

The IW should remain a multiple of 2 MSS segments, to enable efficient ACK compression without incurring unnecessary timeouts.

>> An implementation MUST decrease the IW if more than 95% of connections have IW losses.

Again, this is to ensure an implementation is sufficiently reactive.

>> An implementation MAY group IW values and statistics within subsets of connections. Such grouping MAY use any information about connections to form groups except loss statistics.

There are some TCP connections which might not be counted at all, such as those to/from loopback addresses, or those within the same subnet as that of a local interface (for which congestion control is sometimes disabled anyway). This may also include connections that terminate before the IW is full, i.e., as a separate check at the time of the connection closing.

The period over which the IW is updated is intended to be a long timescale, e.g., a month or so, or 1,000 connections, whichever is longer. An implementation might check the IW once a month, and simply not update the IW or clear the connection counts in months where the number of connections is too small.

5. Discussion

There are numerous parameters to the above algorithm that are compliant with the given requirements; this is intended to allow variation in configuration and implementation while ensuring that all such algorithms are reactive and safe.

This algorithm continues to assume segments because that is the basis of most TCP implementations. It might be useful to consider revising the specifications to allow byte-based congestion given sufficient experience.

The algorithm checks for IW losses only during the first IW after a connection start; it does not check for IW losses elsewhere the IW is used, e.g., during slow-start restarts.

>> An implementation MAY detect IW losses during slow-start restarts in addition to losses during the first IW of a connection. In this case, the implementation MUST count each restart as a "connection" for the purposes of connection counts and periodic rechecking of the IW value.

False positives can occur during some kinds of segment reordering, e.g., that might trigger spurious retransmissions even without a true segment loss. These are not expected to be sufficiently common to dominate the algorithm and its conclusions.

This mechanism does require additional per-connection state which is currently common in some implementations, and is useful for other reasons (e.g., the ISN is used in TCP-AO [RFC5925]). The mechanism also benefits from persistent state kept across reboots, as would be other state sharing mechanisms (e.g., TCP Control Block Sharing [RFC2140]). The mechanism is inspired by RFC 2140's use of information across connections.

The receive window (RWIN) is not involved in this calculation. The size of RWIN is determined by receiver resources, and provides space to accommodate segment reordering. It is not involved with congestion control, which is the focus of this document and its management of the IW.

6. Observations

The IW may not converge to a single, global value. It also may not converge at all, but rather may oscillate by a few MSS as it repeatedly probes the Internet for larger IWs and fails. Both properties are consistent with TCP behavior during each individual connection.

This mechanism assumes that losses during the IW are due to IW size. Persistent errors that drop packets for other reasons - e.g., OS bugs, can cause false positives. Again, this is consistent with TCP's basic assumption that loss is caused by congestion and requires backoff. This algorithm treats the IW of new connections as a long-timescale backoff system.

7. Security Considerations

This algorithm presents an opportunity for an intelligent attack to reduce the IW of a given system, by repeatedly dropping packets during the IW only. An intermediate that can drop packets in a controlled manner can already impact the performance of a connection, and can reduce the congestion window of an ongoing connection in ways that impact performance more than just dropping during the IW.

8. IANA Considerations

This document has no IANA considerations. This section should be removed prior to publication.

9. Conclusions

<Add any conclusions>

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3390] Allman, M., Floyd, S., Partridge, C., "Increasing TCP's Initial Window", RFC 3390 (Standards Track), Oct. 2002.
- [RFC5681] Allman, M., Paxson, V., Blanton, E., "TCP Congestion Control," RFC 5681 (Standards Track), Sep. 2009.

10.2. Informative References

- [Al10] Allman, M., "Initial Congestion Window Specification", (work in progress), draft-allman-tcpm-bump-initcwnd-00, Nov. 2010.
- [Ch10] Chu, J., Dukkupati, N., Cheng, Y., Mathis, M., "Increasing TCP's Initial Window," (work in progress), draft-ietf-tcpm-initcwnd-04, Jun. 2012.
- [Ja88] Jacobson, V., M. Karels, "Congestion Avoidance and Control", Proc. Sigcomm 1988.
- [RFC1323] Jacobson, V., Braden, R., Borman, D., "TCP Extensions for High Performance", RFC 1323, May 1992.

- [RFC2001] Stevens, W., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", RFC2001 (Standards Track), Jan. 1997.
- [RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140 / STD 7 (Informational), Apr. 1997.
- [RFC2414] Allman, M., Floyd, S., Partridge, C., "Increasing TCP's Initial Window", RFC 2414 (Experimental), Sept. 1998.
- [RFC2581] Allman, M., Paxson, V., Stevens, W., "TCP Congestion Control", RFC2581 (Standards Track), Apr. 1999.
- [RFC2861] Handley, M., Padhye, J., Floyd, S., "TCP Congestion Window Validation", RFC2861 (Experimental), June 2000.
- [RFC5925] Touch, J., A. Mankin, R. Bonica, "The TCP Authentication Option", RFC 5925 (Standards Track), June 2010.

11. Acknowledgments

Mark Allman and Aki Nyrjinen contributed to the development of this algorithm. Members of the TCPM mailing list also participated in providing useful feedback.

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

Authors' Addresses

Joe Touch
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90292-6695 U.S.A.

Phone: +1 (310) 448-9151
Email: touch@isi.edu

TCPM Working Group
Internet Draft
Intended status: Informational
Expires: April 2012

J. Touch
USC/ISI
October 24, 2011

Shared Use of Experimental TCP Options
draft-touch-tcpm-experimental-options-00.txt

Status of this Memo

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

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

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

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

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

This Internet-Draft will expire on April 24, 2012.

Copyright Notice

Copyright (c) 2011 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.

Abstract

This document describes how TCP option codepoints can support concurrent experiments. The suggested mechanism avoids the need for a coordinated registry, and is backward-compatible with currently known uses.

Table of Contents

1. Introduction.....	2
2. Conventions used in this document.....	3
3. TCP Experimental Option Structure.....	3
4. Security Considerations.....	4
5. IANA Considerations.....	5
6. References.....	5
6.1. Normative References.....	5
6.2. Informative References.....	5
7. Acknowledgments.....	6

1. Introduction

TCP includes options to enable new protocol capabilities that can be activated only where needed and supported [RFC793]. The space for identifying such options is small – 256 values, of which 31 are assigned at the time this document was published [IANA]. Two of these codepoints are allocated to support experiments (253, 254) [RFC4727]. These numbers are intended for testing purposes, and implementations need to assume they can be used for other purposes, but this is often not the case.

There is no mechanism to support shared use of the experimental option codepoints. Experimental options 245 and 255 are deployed in operational code to support an early version of TCP authentication. Option 253 is also documented for the experimental TCP Cookie Transaction option [RFC6013]. This shared use results in collisions in which a single codepoint can appear multiple times in a single TCP segment and each use is ambiguous.

Other options have been used without assignment, notably 31–32 (TCP cookie transactions, as originally distributed and in its API doc) and 76–78 (tcpcrypt) [Bill][Sill]. Commercial products reportedly also use unassigned options 33 and 76–78 as well.

There are a variety of proposed approaches to address this issue. The first is to relax the requirements for assignment of TCP

options, allowing them to be assigned more readily for protocols that have not been standardized through the IETF process [RFC5226]. A second would be to assign a larger pool to options, and to manage their sharing through IANA coordination [Ed11].

This document proposes a solution that does not require additional codepoints and also avoids IANA participation. A short nonce is added to the structure of the experimental TCP option structure. The nonce helps reduce the probability of collision of independent experimental uses of the same option codepoint. This feature increases the size of experimental options, but the size can be reduced when the experiment is converted to a standard protocol with a conventional codepoint assignment.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying RFC-2119 significance.

In this document, the characters ">>" preceding an indented line(s) indicates a compliance requirement statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the explicit compliance requirements of this RFC.

3. TCP Experimental Option Structure

TCP options have the current common structure, where the first byte is the codepoint (Kind) and the second is the length of the option in bytes (Length):

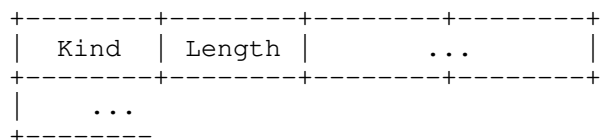


Figure 1 TCP Option Structure [RFC793]

This document extends the option structure for experimental codepoints (253, 254) as follows:

Kind	Length	Nonce
Nonce	...	

Figure 2 TCP Experimental Option with a Nonce

>> Protocols using the TCP experimental option codepoints (253, 254) SHOULD use nonces as described in this document.

The nonce is selected by the protocol designer when the experimental option is defined. The Nonce is selected any of a variety of ways, e.g., using the Unix time() command or bits selected by an arbitrary function (such as a hash).

>> The nonce SHOULD be selected to reduce the probability of collision.

The length of the nonce is intended to be 32 bit in network standard byte order. It can be shorter if desired (e.g., 16 bits), with a corresponding increased probability of collision and thus false positives.

During TCP processing, experimental options are matched against both the experimental codepoints and the Nonce value for each implemented protocol.

>> Experimental options that have nonces that do not match implemented protocols MUST be ignored.

The remainder of the option is specified by the particular experimental protocol.

Use of a nonce uses additional space in the TCP header and requires additional protocol processing by experimental protocols. Because these are experiments, neither consideration is a substantial impediment; a finalized protocol can avoid both issues with the assignment of a dedicated option codepoint later.

4. Security Considerations

The mechanism described in this document is not intended to provide security for TCP option processing. False positives are always possible, where a Nonce matches a legacy use of these options or a protocol that does not implement the mechanism described in this document.

>> Protocols that are not robust to such false positives SHOULD implement other measures to ensure they process options for their protocol only, such as checksums or digital signatures among cooperating parties of their protocol.

5. IANA Considerations

This document has no IANA considerations. This section should be removed prior to publication.

6. References

6.1. Normative References

- [RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, Sep. 1981.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4727] Fenner, B., "Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers", RFC 4727, Nov. 2006.

6.2. Informative References

- [Bil1] Bittau, A., D. Boneh, M. Hamburg, M. Handley, D. Mazieres, Q. Slack, "Cryptographic protection of TCP Streams (tcpcrypt)", work in progress, draft-bittau-tcp-crypt-01, Aug. 29, 2011.
- [Ed1] Eddy, W., "Additional TCP Experimental-Use Options", work in progress, draft-eddy-tcpm-addl-exp-options-00, Aug. 16, 2011.
- [IANA] IANA web pages, <http://www.iana.org/>
- [RFC5226] Narten, T., H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC6013] Simpson, W., "TCP Cookie Transactions (TCPCT)", RFC 6013, Jan. 2011.
- [Si1] Simpson, W., "TCP Cookie Transactions (TCPCT) Sockets Application Program Interface (API)", work in progress, draft-simpson-tcpct-api-04, Apr. 7, 2011.

7. Acknowledgments

This document was motivated by discussions on the IETF TCPM mailing list and by Wes Eddy's proposal [Ed11].

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

Authors' Addresses

Joe Touch
USC/ISI
4676 Admiralty Way
Marina del Rey, CA 90292-6695 U.S.A.

Phone: +1 (310) 448-9151
Email: touch@isi.edu

