TCP Maintenance and Minor Extensions
Internet-Draft
Intended status: Experimental
Expires: 8 January 2021
7 July 2020

M. Amend
DT
J. Kang
Huawei

Multipath TCP Extension for Robust Session Establishment
draft-amend-tcpm-mptcp-robe-00

Abstract

Multipath TCP extends the plain, single-path limited, TCP towards the capability of multipath transmission. This greatly improves the reliability and performance of TCP communication. For backwards compatibility reasons the Multipath TCP was designed to setup successfully an initial path first, after which subsequent paths can be added for multipath transmission. For that reason the Multipath TCP has the same limitations as the plain TCP during connection setup, in case the selected path is not functional.

This document proposes a set of implementations and possible combinations thereof, that provide a more Robust Establishment (RobE) of MPTCP sessions. It includes RobE_TIMER, RobE_SIM, RobE_eSIM and RobE_IPS.

RobE_TIMER is designed to stay close to MPTCP in that standard functionality is used wherever possible. Resiliency against network outages is achieved by modifying the SYN retransmission timer: If one path is defective, another path is used.

RobE_SIM and RobE_eSIM provides the ability to simultaneously use multiple paths for connection setup. They ensure connectivity if at least one functional path out of a bunch of paths is given and offers beside that the opportunity to significantly improve loading times of Internet services.

RobE_IPS provides a heuristic to select properly an initial path for connection establishment with a remote host based on empirical data derived from previous connection information.

In practice, these independent solutions can be complementary used. This document also presents the design and protocol procedure for those combinations in addition to the respective stand-alone solutions.

Status of This Memo

This Internet-Draft will expire on 8 January 2021.

Copyright Notice

Table of Contents

1. Introduction

Multipath TCP Robust Session Establishment (MPTCP RobE) is a set of extensions to regular MPTCP [RFC6824] and its next version [RFC8684], which releases single path limitations during the initial connection setup. Several scenarios require and benefit from a reliable and in time connection setup which is not covered by [RFC6824] and [RFC8684] so far. MPTCP was designed to be compliant with the TCP standard [RFC0793] and introduced therefore the concept of an initial TCP flow while adding subsequent flows after successful multipath negotiation on the initial path. While fulfilling its purpose, MPTCP is however fully dependent on the transmission characteristics of the communication link selected for initiating MPTCP.

Figure 1 shows the traditional way of MPTCP handshaking with a MP_CAPABLE exchanged first, followed when successful negotiated by additional flows engaging MP_JOIN. [RFC6824] and the next MPTCP [RFC8684] differ in that a Key-A is sent with the first MP_CAPABLE or not.

```
               Host A                                  Host B
        ------------------------                       ----------
        Address A1    Address A2                       Address B1
        ----------    ----------                       ----------
            |             |                                |
            |            SYN + MP_CAPABLE(Key-A[*])        |
            |---------------------------------------------->|
            |<----------------------------------------------|
            |          SYN/ACK + MP_CAPABLE(Key-B)         |
            |             |                                |
            |        ACK + MP_CAPABLE(Key-A, Key-B)        |
            |---------------------------------------------->|
            |             |                                |
            |             |   SYN + MP_JOIN(Token-B, R-A)  |
            |             |------------------------------->|
            |             |<-------------------------------|
            |             | SYN/ACK + MP_JOIN(HMAC-B, R-B) |
            |             |                                |
            |             |     ACK + MP_JOIN(HMAC-A)      |
            |             |------------------------------->|
            |             |<-------------------------------|
            |             |             ACK                |

            [*] Key-A in the first MP-capable is related to
                RFC6824 only and does not exist in RFC8684.
```

Figure 1: MPTCP connection setup

Multipath TCP itself enables hosts to exchange packets belonging to a single connection over several paths. Implemented in mobile phones (UEs), these paths are usually assigned to different network interfaces within the UE and corresponds to different networks such as cellular and WiFi. The path or network interface for initiating the initial subflow setup is most often provided by the operation system of the UE. For example, if a cellular connection and WiFi are present in a mobile phone, WiFi is usually the interface offered to initiate the MPTCP session.

This design falls short in situations where the default path does not provide the best performance compared to other available paths. In a worst case the default path is not even capable of setting up the initial flow letting any other functional path unused. For example, if the WiFi signal is weak, broken or cannot forward traffic to the destination, the establishment of the subflow will be delayed or impossible. This in turn, leads to a longer startup delay or no communication at all for services using MPTCP even if other functional paths are available. Even in scenarios where all paths are functional but services would benefit from a setup over the path with the lowest latency, MPTCP has no mean to support this demand.

It can be concluded, that sequential path establishment relying with an initial path establishment over an external given default route will result in experience reduction when using MPTCP. So this document proposes solutions to overcome the aforementioned limitations and provides a more robust connection setup compared to traditional MPTCP.

RobE_SIM and RobE_eSIM aims to overcome the limitations of [RFC6824] and [RFC8684], using one initial flow and introduces the concept of potential initial flows triggered simultaneously. Potential initial flows gives the freedom to use more than one path to request multipath capability and select the initial flow at a later point. Potential initial flow mechanisms and the gain of robustness and performance over the traditional MPTCP connection setup are evaluated in [RobE_slides] and [RobE_paper]. RobE_SIM is a break-before-make mechanism, guaranteeing at least the robust connection establishment, however the RobE_eSIM reuses every potential initial flow request to combine it with less overhead and accelerated multipath availability, leveraging a new MPTCP option MP_JOIN_CAP. From a standardization perspective, the RobE_SIM is fully compliant with [RFC6824] and [RFC8684] and is herein more of a descriptive and procedural nature. The RobE_eSIM requires a new MPTCP option but with the potential to significantly improve the MPTCP experience.

For the limitation of the default initial path, RobE_IPS makes no changes to standard MPTCP procedure and improves the performance of connection establishment by introducing an initial path selection strategy and required algorithms. The input for strategy and algorithms is the transmission status information which represents the transmission performance of each available path or network interface. The transmission status information is characterized by at least one of the parameters: signal strength, throughput, round-trip time (RTT) and link success rate. In this way, a path with better transmission performance can be learned and determined and the respective network interface can be used for connection establishment.

The most simple approach for a robust MPTCP session establishment is RobE_TIMER, iterating the process of initial path establishment over all available paths, if the previous try has failed. Triggering a new try on a next path is depending on an expiration timer, preferably re-use TCP's in-built expiration timer.

Table 1 summarizes the impact of RobE_TIMER, RobE_SIM, RobE_eSIM and RobE_IPS compared to [RFC6824] and [RFC8684].

Scenario	MPTCP	RobE_TIMER	RobE_SIM	RobE_eSIM	RobE_IPS
IP packet loss	Delayed connection	In the scope of timer	No impact	No impact	Delayed connection
IP broken	No connection	In the scope of timer	No impact	No impact	No connection
IP setup duration dependency	Default route	Default route (+ path 1..n)	Fastest path	Fastest path	Selected path
MP availability duration	MP_CAPABLE HS + MP_JOIN HS	sum_1..n(MP_CAPABLE_n HS) + MP_JOIN HS	MP_CAPABLE HS + MP_JOIN HS	max(MP_CAPABLE_1 .. MP_CAPABLE_n HS)	MP_CAPABLE HS + MP_JOIN HS
Guaranteeing session setup	Depends on the default route	Yes	Yes	Yes	Depends on selection

Table 1: Overview RobE features during initial connection setup

> IP: Initial Path; MP: Multi-Path; HS: Handshake

1.1. Terminology

This document makes use of a number of terms that are either MPTCP-specific or have defined meaning in the context of MPTCP, as follows:

Path: A sequence of links between a sender and a receiver, defined in this context by a 4-tuple of source and destination address/port pairs.

Subflow: A flow of TCP segments operating over an individual path, which forms part of a larger MPTCP connection. A subflow is started and terminated similar to a regular TCP connection.

2. Implementation without MPTCP protocol adaptation

RobE_TIMER, RobE_SIM and RobE_IPS are compatible with the current MPTCP protocol definitions in [RFC6824] and [RFC8684] but may be lack of the full optimization potential which require protocol adaptation in Section 3. Following sections will describe them in detail.

2.1. Re-transmission Timer(RobE_TIMER)

In RobE_TIMER, a new connection is initiated by sending a SYN+MP_CAPABLE along the initial path. If this path is functional, the solution will perform identical to classic MPTCP: the initial flow will be established, and subsequent flows can be created afterwards. If however the initial path is faulty, the retransmission will be triggered on another path. This path might circumvent the dysfunctional network, and allow the client to create an initial subflow. The first path is now seen as a subsequent path and the client sends SYN+MP_JOIN messages to create a subsequent flow.

In high latency networks, the initial SYN+MP_CAPABLE might be delayed until the client retries on another path. Once the second SYN arrives at the server, it will try to complete the three-way handshake. If the first SYN was delayed by more than the retransmission time plus half a Round Trip Time (RTT) of the second path, it will arrive at the server after the second SYN. The server could now treat the segment as obsolete and drop it.

```
                Host A                                    Host B
         ------------------------                       ----------
         Address A1    Address A2                       Address B1
         ----------    ----------                       ----------
             |             |                                |
             |            SYN + MP_CAPABLE(Key-A[*])        |
             |Timer---------------------------------------->|
             |             |   SYN + MP_CAPABLE(Key-A’[*])  |
             |             |------------------------------->|
             |             |   SYN/ACK+MP_CAPABLE(Key-B’)   |
             |             |<-------------------------------|
             |             | ACK + MP_CAPABLE(Key-A’,Key-B’)|
             |             |------------------------------->|
             |             |   SYN + MP_JOIN(Token-B’,R-A)  |
             |--------------------------------------------->|
             |   Subflow will be set up as normal MPTCP     |
             |                                              |

              [*] Key-A in the first MP-capable is related to
               RFC6824 only and does not exist in RFC8684.
```

Figure 2: The Robe_TIMER Solution

Immediately after sending the final ACK of the initial handshake, subflows are established on the remaining paths as defined in [RFC6824] and [RFC8684]

[Notes: How to set the Timer is TBD. If there is the case that the first SYN on default path arrives earlier than that from the second path, the MPTCP connection will be initialized on the path of the first SYN. The server could treat the second SYN as obsolete and drop it.]

2.2. Simultaneous Initial Paths Simple Version (RobE_SIM)

RobE_SIM is a sender only implementation and no negotiation is required. In RobE_SIM, the MPTCP connection setup benefits from the fastest path. As shown in Figure 3, host A initiates the connection handshake on more than one path independently (SA1 and SA2). The paths selected for RobE_SIM and referred to as potential initial flows, can belong to the number of interfaces on the device or a subset selected on experience. When Host A receives the first SYN/ACK back from Host B (SA3), the path carrying this message is identified as the normal initial path. Host A sends then immediately a TCP RST message (SA6.1) on any other path used for simultaneous connection setup causing an immediate termination of assigned flows (break-before-make). The terminated ones are merged as subsequent subflows following the JOIN procedure described in [RFC6824] and [RFC8684]. The process is equivalent to any other scenario where the SYN/ACK arrives on an other path than depicted in Figure 3.

```
                   Host A                                  Host B
           ------------------------                       ----------
           Address A1    Address A2                       Address B1
           ----------    ----------                       ----------
               |             |                                |
               |            SYN + MP_CAPABLE(Key-A[*])        |
       (SA1)   |--------------------------------------------->|  (SB1)
               |             |      SYN + MP_CAPABLE(Key-A'[*]) |
       (SA2)   |             |------------------------------->|  (SB2)
               |             |                                |
       (SA3)   |<---------------------------------------------|  (SB3)
               |          SYN/ACK + MP_CAPABLE(Key-B)         |
       (SA4)   |             |<-------------------------------|  (SB4)
               |             |   SYN/ACK + MP_CAPABLE(Key-B') |
               |             |                                |
               |        ACK + MP_CAPABLE(Key-A, Key-B)        |
       (SA5)   |--------------------------------------------->|  (SB5)
               |             |             RST                |
       (SA6.1) |             |------------------------------->|  (SB6.1)
      RobE SIM |             |   SYN + MP_JOIN(Token-B, R-A)  |
      (robust) |             |------------------------------->|
               |             |         MP_JOIN Process...     |

               [*] Key-A in the first MP-capable is related to
                   RFC6824 only and does not exist in RFC8684.
```

Figure 3: MPTCP RobE_SIM Connection Setup

2.3. Heuristic Initial Path Selection (RobE_IPS)

2.3.1. Architecture

Figure 4 provides the architecture for RobE_IPS and employs an "Initial Path Selection" logic which can be integrated into the MPTCP stack or exists as an isolated module in the terminal. The IPS logic has access to a set of transmission status information for each available path or its belonging network interfaces. When an application starts a first communication, IPS selects based on the available path transmission characteristics the path with the highest probability to succeed.

```
         +-------------------+              +-------------------+
         |     Terminal      |              |      Server       |
         |  +-------------+  |              |  +-------------+  |
         |  |Application n|  |              |  |Application n|  |
         |  +-------------+  |              |  +-------------+  |
         |        |          |              |         |         |
         |  +-------------+  |              |         |         |
         |  |Initial-path |  |-------+      |         |         |
         |  |  Selection  |  |       |      |         |         |
         |  +-------------+  |       |      |         |         |
         |        |          |  +--------+  |         |         |
         |  +-------------+  |--|Internet|--|  +-------------+  |
         |  | MPTCP Stack |  |--+--------+--|  | MPTCP Stack |  |
         |  +-------------+  |              |  +-------------+  |
         +-------------------+              +-------------------+
```

Figure 4: Architecture for Initial-path Selection

2.3.2. Typical Scenarios

Two typical RobE_IPS scenarios are presented in this section. Figure 5 shows that the "Initial Path Selection" logic executed for each MPTCP connection establishment. On the other hand Figure 6 describes that "Initial Path Selection" in case no path information are available. Considering the fact that no heuristics are given before a recent MPTCP connection was established, the default initial path can be adopted. Further combinations and implementations with more or less sophisticated heuristics are possible.

```
                           +---------------+
                           |  Application  |
                           |    Request    |
                           +---------------+
                                   |
                                   V
                           +---------------+
                      +--->| Initial-path  |<---+
                      |    |   Selection   |    |
                      |    +---------------+    |
                      |            |            |
                      |            V            |Info
                      |    +---------------+    |
                      |    |  Set initial  |----+
                      |    |     path      |
                      |    |   for MPTCP   |
                      |    +---------------+
                      |            |
                      |            V
                      |    +---------------+
                      |No  |Establish MPTCP|
                      +----|  Connection   |
                           +---------------+
                                   |Yes
                                   V
```

Figure 5: RobE_IPS for each connection establishment

```
                        +---------------+
                        |  Application  |
                        |    Request    |
                        +---------------+
                                 |
                                 V
                        +---------------+Yes
                        |     First     |------------+
                        |  Connection?  |            |
                        +---------------+            |
                                 |No                 |
                                 V                   |
                        +---------------+            V
                 +----->| Initial-path  |<-+     +-------+
                 |      |   Selection   |  |     |Default|
                 |      +---------------+  |     |initial|
                 |               |         |     |  path |
                 |               |         |     +-------+
                 |               V         |Info     |
                 |      +---------------+  |         |
                 |      |  Set initial  |--+         |
                 |      |     path      |            |
                 |      |   for MPTCP   |            |
                 |      +---------------+            |
                 |               |                   |
                 |               V                   |
                 |No    +---------------+            |
                 +------| Successful?  |<-----------+
                        +---------------+
                                 |Yes
                                 V
```

Figure 6: RobE_IPS using default route when no meaningful heuristic available

Figure 7 shows the process flow of "Initial Path Selection". Upon a request from an application, the IPS logic will acquire transmission status information which represents the transmission performance of each available path or network interface and evaluate it. The transmission status information is characterized by at least one of the parameters: signal strength, throughput, round-trip time (RTT) and link success rate. In this way, the path with the best transmission performance can be determined and used for connection establishment.

```
                                     |
                                     V
                       +---------------------------+
                       |Acquire transmission status|
                       | info for available paths  |
                       +---------------------------+
                                     |
                                     V
                       +---------------------------+
                       |   Evaluating the status   |
                       |    for available paths    |
                       +---------------------------+
                                     |No
                                     V
                       +---------------------------+
                       | Determining an available  |
                       |     path with better      |
                       |      transmission         |
                       |      performance          |
                       +---------------------------+
                                     |
                                     V
                       +---------------------------+
                       |     Using the network     |
                       |         interface         |
                       |corresponding to the path |
                       | with better transmission  |
                       |performance for connection |
                       |       establishment       |
                       +---------------------------+
                                     |
                                     V
```

Figure 7: Implementation process for Initial Path Selection

2.3.3. Path decision information

The level of heuristic can be mainly divided into three layer: application level, transport-layer level and link-layer level based on the information acquisition method. For example, RTT can be calculated for each path within a MPTCP connection and belongs thereof to the transport-layer level. The transmission status information for each available path SHOULD be characterized by at least one of the parameters: signal strength, throughput, RTT and link success rate. Application level information are more seen for statistical purposes.

- Application level: application name, domain name, port number and location.

- Transport-layer level: RTT, CWND, Error rate.

2.3.4. Initial Path Selection use local RTT information

Figure 8 presents a "Initial Path Selection" logic based on RTT, e.g. assuming two paths over LTE and WiFi access. RTT calculation on the transport layer usually reflect the time when an information is sent and an related acknowledgment received. For an asymmetric usage (e.g. download only) of a communication it might happen that recent RTT calculation is only available on sender side which is possibly not the side which employs the IPS logic. A solution for this can be found in Section 3.2. Instead of using the most recent RTT value of a path a filtered value consisting of several measured RTTs can be used. A RTT can also be derived from link layer information but may has a limited meaning when it does not picture the end-to-end latency.

```
                +-------------------+
                |    New Session    |
                +-------------------+
                          |
                          V
                +-------------------+ No
                |Running Connections|-----------+
                |(LTE.RTT<WiFi.RTT) |           |
                +-------------------+           |
                          |Yes                  |
                          V                     V
                +-------------------+   +-----------------+
                |     Set LTE as    |   |   Set WiFi as   |
                |    initial path   |   |  initial path   |
                +-------------------+   +-----------------+
```

Figure 8: Initial-path Selection based on RTT

2.4. Combination of RobE_SIM and RobE_IPS

In an implementation, a single solution may not be sufficient to get an expected behavior. Combination of approaches to improve robustness is recommended therefore. Figure 9 shows the combination of RobE_SIM and RobE_IPS. RobE_SIM can be used at the very beginning when the sender is absent of path information followed by RobE_IPS for consecutive connections.

```
                         +--------------+
                         |  Application |
                         |    Request   |
                         +--------------+
                                 |
                                 V
                         +--------------+
                  +----->| Any data for | No
                  |      | Initial Path |----------+
                  |      |  Selection?  |          |
                  |      +--------------+          |
                  |              |                 |
                  |              V                 V
                  |      +--------------+     +--------+
                  |      | Initial-path |     |RobE_SIM|
                  |      |  Selection   |<-+  +--------+
                  |      +--------------+  |       |
                  |              |         |       |
                  |              V         |Info   |
                  |      +---------------+ |       |
                  |No    |Establish MPTCP|-+       |
                  +------|  Connection   |<--------+
                         +---------------+
                                 |
                                 V
                   No    +---------------+
                  <------|  Successful ? |
                  Network+---------------+
                  Problem        |Yes
                                 V
```

Figure 9: Combination of RobE_SIM and RobE_IPS

2.5. Combination of RobE_TIMER and RobE_IPS

Since RobE_IPS solely does not guarantee that session can be set up on the selection of initial path, it can also be combined with RobE_TIMER which generates less overhead compared to the combination with RobE_SIM in Section 2.4 and guarantees session setup. RobE_TIMER can be introduced to optimize the control of path switching when the initial path selected by RobE_IPS is dysfunctional. When the system enables RobE_IPS and uses the selected initial path for session establishment,it sets the timer for path switching. When timer is expired, the system will change to another path to re-establish connection according to Section 2.1.

```
                                +---------------+
                                |  Application  |
                                |    Request    |
                                +---------------+
                                        |
                                        V
                                +---------------+
                                |  Initial Path |
                         |----->|   Selection   |
                         |      | and Set Timer |
                         |      +---------------+
                         |              |
                         |              V
                         |Yes   +---------------+
                         +------| Timer is up?  |
                                +---------------+
                                        |No
                                        V
                                +---------------+
                                |Establish MPTCP|
                                |  Connection   |
                                +---------------+
                                        |
                                        V
                          No    +---------------+
                         <------|  Successful?  |
                         Network+---------------+
                         Problem        |Yes
                                        V
```

Figure 10: Combination of RobE_Timer and RobE_IPS

3. Implementation with Bi-directional MPTCP Support

Solutions which requires bi-directional support between two MPTCP hosts promise to have better and possibly more features. However, they cannot be defined without extending current standards in [RFC6824] and [RFC8684]. The RobE_SIM and RobE_IPS approach are both capable of profit from an explicit support of the remote end host and defined within this section.

3.1. Simultaneous Initial Paths Extended Version (RobE_eSIM)

RobE_eSIM extends RobE_SIM by reusing the potential initial flows. This eliminates the overhead from RobE_SIM by introducing a new option MP_JOIN_CAP and accelerate the transmission speed by early availability of multiple paths. Further it relaxes the dependency on a reliable third ACK of the 3-way handshake in [RFC8684]. Remote endpoint support can be negotiated in two ways, an implicit in Section 3.1.1 or explicit in Section 3.1.2.

3.1.1. RobE_eSIM implicit Negotiation and Procedure

Similar to RobE_SIM in Section 2.2, the establishment process of [RFC6824] or [RFC8684] is applied independently on multiple path simultaneously. In Figure 11 this is shown in SA1 and SA2. The first path which returns a SYN/ACK (e.g. SA3) is selected as the initial path and proceed with the traditional establishment process (SA5). Any other path which has to send the final ACK of the 3-way handshake includes a new option MP_JOIN_CAP (see definition in Section 3.1.3.2) instead of a MP_CAPABLE (SA6.2).

```
                   Host A                                  Host B
           ------------------------                       ----------
           Address A1    Address A2                       Address B1
           ----------    ----------                       ----------
               |             |                                |
               |            SYN + MP_CAPABLE(Key-A[*])        |
       (SA1)   |---------------------------------------------->|  (SB1)
               |             |    SYN + MP_CAPABLE(Key-A'[*]) |
       (SA2)   |             |------------------------------->|  (SB2)
               |             |                                |
       (SA3)   |<---------------------------------------------|  (SB3)
               |          SYN/ACK + MP_CAPABLE(Key-B)         |
       (SA4)   |             |<-------------------------------|  (SB4)
               |             |   SYN/ACK + MP_CAPABLE(Key-B') |
               |             |                                |
               |        ACK + MP_CAPABLE(Key-A, Key-B)        |
       (SA5)   |---------------------------------------------->|  (SB5)
               |             |                                |
       (SA6.2) |             |                                |  (SB6.2)
      RobE EXT |             | ACK + MP_JOIN_CAP(Key-A, HMAC) |
      (+fast)  |             |------------------------------->|

               [*] Key-A in the first MP-capable is related to
                   RFC6824 only and does not exist in RFC8684.
```

Figure 11: MPTCP RobE_eSIM implicit Connection Setup

Following the possible process in Figure 11, two further constellations are imaginable and elaborated below.

1. In the flow diagram Figure 11, A1<->B1 is assumed to be the initial flow. A2<->B1 shall be recycled and the ACK is sent with MP_JOIN_CAP. Furthermore, the MP_CAPABLE arrives first at Host B (SB5) and the MP_JOIN_CAP afterwards (SB6.2). When the MP_JOIN_CAP is received, Host B has to iterate over the connection list once (like MP_JOIN) and check for Key-A availability. If a Key-A connection is found, this one is validated against the HMAC value. The validation has two reasons: first, several Key-A can exist, because different hosts may choose the same Key-A by accident. Furthermore, no one can join a connection by just recording/brute-forcing Key-A and duplicating the request.

2. Like above, but MP_JOIN_CAP arrives before last MP_CAPABLE at Host B

   * [RFC8684]; Based on Key-A, Host B will iterate over the connection list, but it will not find a match, because Key-A of the previous selected initial flow (SA3, SA5) has not arrived yet. So it will continue with a fast iteration only over the connections which are still in establishment phase using the 10 bit Key-B fast hash (crc16(Key-B) & 0x3FF). If it matches against a (precomputed) existing Key-B_fast_hash in the connection list, it will validate the request using the HMAC(Key-A+B+B’) to ensure legitimation. If successful, both, the initial flow and the MP_JOIN_CAP flow, can be immediately established. This is true, because without the knowledge of Key-B, Host A could not calculate the HMAC. So it is clear, that Host A had received the SYN/ACK (SB3). This also mitigates the exchange of a reliable ACK during the handshake process. MPTCP sends the Key-A only with the last ACK and therefore prevents subsequent flow establishment until successful reception at Host B. Using RobE_EXT, the reception of a MP_JOIN_CAP ([RFC8684]) is sufficient to establish both, the path carrying Key-B and Key-B’.

   * [RFC6824]; Can match based on Key-A, same effort as for a MP JOIN.

3. A2<->B1 is selected as initial flow, because the respective SYN/ACK returns earlier at Host A. It is the same as above, just the other way round.

3.1.2. RobE_eSIM explicit Negotiation and Procedure

The process of an explicit negotiation of RobE_eSIM follows Figure 11 but uses the ROBE_eSIM_EN option Figure 13 additionally during the handshake procedure.

```
               Host A                                        Host B
      ------------------------                             ----------
      Address A1    Address A2                             Address B1
      ----------    ----------                             ----------
          |             |                                        |
          |      SYN+MP_CAPABLE+ROBE_eSIM_EN(Key-A[*])           |
          |----------------------------------------------------->|
          |             | SYN+MP_CAPABLE+ROBE_eSIM_EN(Key-A'[*]) |
          |             |--------------------------------------->|
          |      SYN/ACK+MP_CAPABLE+ROBE_eSIM_EN(Key-B)          |
          |<---------------------------------------------------->|
          |             | SYN/ACK+MP_CAPABLE+ROBE_eSIM_EN(Key-B')|
          |             |<---------------------------------------|
          |      ACK+MP_CAPABLE(Key-A,Key-B)                     |
          |----------------------------------------------------->|
          |             |                                        |
          |             |  ACK+MP_JOIN_CAP(Key-A,HMAC)           |
          |             |--------------------------------------->|
          |             |                                        |

            [*] Key-A in the first MP-capable is related to
                RFC6824 only and does not exist in RFC8684.
```

Figure 12: MPTCP RobE_eSIM explicit Connection Setup

3.1.3. Protocol Adaptation

3.1.3.1. ROBE_eSIM_EN Option

```
                        1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +---------------+---------------+-------+-------+---------------+
     |     Kind      |    Length     |Subtype|    (reserved)         |
     +---------------+---------------+-------+-------+---------------+
```

Figure 13: ROBE_eSIM_EN_OPTION

3.1.3.2. MP_JOIN_CAP Option

```
                        1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +---------------+---------------+-------+-------+---------------+
   |     Kind      |    Length     |Subtype|       |    ADDR_ID    |
   +---------------+---------------+-------+-------+---------------+
   |                    Sender's Key-A (64 bits)                   |
   |                                                               |
   |                                                               |
   +---------------------------------------------------------------+
   |                        HMAC (>=96 bits)                       |
   |                                                               |
   |                                                               |
   :                                                               :
   +---------------------------------------------------------------+
    Key-B_fast_hash = crc16(Key-B) & 0x3FF         -> (10bit)
    HMAC_keys =  HMAC(Key-A+Key-B+Key-B')          -> (>=96bit)
    HMAC =  (HMAC_keys & ˜0x3FF) | Key-B_fast_hash  -> (size HMAC_keys)
```

Figure 14: MP_JOIN_CAP

Computational effort on receiver side is most often expected to be the same as with MP_JOIN. Key-A ensures identification of related flows Key-B_fast_hash enables MP session even when selected initial flow is not fully established yet (slight computational overhead). HMAC authenticates relationship of initial and potential initial flows.

3.1.4. Fallback Mechanisms

3.1.4.1. Fallback mechanism for implicit RobE_eSIM

[TBD]

3.1.4.2. Fallback mechanism for explicit RobE_eSIM

This mechanism considers that both sides support MPTCP capability but the receiver does not equipped with RobE_eSIM. MPTCP session with RobE_eSIM negotiation will seamlessly fallback to normal MPTCP process.

[Requires further check how an unaware Host B reacts on possible ROBE_eSIM_EN; Ignore or RST? See also RFC6824 Sec. 3.6 "Should fallback [...] the path does not support the MPTCP options"]

```
                    Host A                              Host B
          ------------------------                    ----------
          Address A1    Address A2                    Address B1
          ----------    ----------                    ----------
              |             |                             |
              |      SYN+MP_CAPABLE+ROBE_eSIM_EN          |
              |------------------------------------------>|
              |             | SYN+MP_CAPABLE+ROBE_eSIM_EN |
              |             |---------------------------->|
              |        SYN/ACK+MP_CAPABLE                 |
              |<----------------------------------------->|
              |             |    SYN/ACK+MP_CAPABLE       |
              |             |<----------------------------|
              |       ACK+MP_CAPABLE                      |
              |------------------------------------------>|
              |             |          RST                |
              |             |---------------------------->|
              |             |       SYN+MP_JOIN           |
              |             |---------------------------->|
              |             |     MP_JOIN Process...      |
              |             |                             |
```

Figure 15: Fallback to MPTCP when missing RobE_eSIM support

3.1.4.3. Fallback to regular TCP when missing MPTCP support

When the receiver is not MPTCP enabled, MPTCP session with RobE_eSIM negotiation will seamlessly fallback to regular process which is illustrated in this section.

```
                    Host A                              Host B
          ------------------------                    ----------
          Address A1    Address A2                    Address B1
          ----------    ----------                    ----------
              |             |                             |
              |      SYN+MP_CAPABLE+ROBE_eSIM_EN          |
              |------------------------------------------>|
              |             | SYN+MP_CAPABLE+ROBE_eSIM_EN |
              |             |---------------------------->|
              |         SYN/ACK                           |
              |<----------------------------------------->|
              |             |    SYN/ACK                  |
              |             |<----------------------------|
              |           ACK                             |
              |------------------------------------------>|
              |             |          RST                |
              |             |---------------------------->|
              |             |   Regular TCP Process...    |
              |             |                             |
```

Figure 16: Fallback to TCP without MPTCP support

3.1.5. Comparison Robe_SIM and RobE_eSIM

Potential initial flows in RobE_SIM Section 2.2 and RobE_eSIM Section 3.1 guarantee MPTCP session establishment if at least one selected path for session establishment is functional. Figure 17 makes the differences between both approaches visible and points to the latest decision possibility during session setup when RobE_SIM or RobE_eSIM can be selected. Until SA5 in Figure 17 traditional MPTCP connection setup is independently applied on multiple paths simultaneously and offers to select the initial flow later (potential initial flows). The final decision which path is selected as the main one and the handling of the remaining flow(s) differs in SA6.1 when RobE_SIM is applied or instead SA6.2 RobE_eSIM.

```
                Host A                                  Host B
        ------------------------                       ----------
        Address A1    Address A2                       Address B1
        ----------    ----------                       ----------
            |             |                                |
            |            SYN + MP_CAPABLE(Key-A[*])        |
    (SA1)   |--------------------------------------------->|  (SB1)
            |             |    SYN + MP_CAPABLE(Key-A’[*]) |
    (SA2)   |             |------------------------------->|  (SB2)
            |             |                                |
    (SA3)   |<---------------------------------------------|  (SB3)
            |          SYN/ACK + MP_CAPABLE(Key-B)         |
    (SA4)   |             |<-------------------------------|  (SB4)
            |             |   SYN/ACK + MP_CAPABLE(Key-B’) |
            |             |                                |
            |        ACK + MP_CAPABLE(Key-A, Key-B)        |
    (SA5)   |--------------------------------------------->|  (SB5)
            |             |             RST                |
    (SA6.1) |             |------------------------------->|  (SB6.1)
   RobE SIM |             |                                |
   (robust) |             |                                |
   --------------------------------------------------------------------
   RobE EXT |             |                                |
   (+fast)  |             | ACK + MP_JOIN_CAP(Key-A, HMAC) |
    (SA6.2) |             |------------------------------->|  (SB6.2)

            [*] Key-A in the first MP-capable is related to
                RFC6824 only and does not exist in RFC8684.
```

Figure 17: MPTCP RobE_SIM and RobE_eSIM connection setup

3.1.6. Security Consideration

[Tbd, however no differences to [RFC6824] and {{RFC8684} are expected]

3.2. Heuristic Initial Path Selection with remote RTT Measurement

3.2.1. Description

Usually the path RTT can be determined by a time difference between sending a package and an ACK and is integrated into the TCP protocol. For asymmetric transmission, the latest RTT for TCP flows is calculated by the side which sends data at latest and possible does not correspond to the site which employs RobE_IPS. This problem is already elaborated in Section 2.3.4 and can be solved by transmitting the RTT information per subflow. The negotiation procedure is depicted in Figure 18 and uses the MPTCP option L_RTT_EN defined in Section 3.2.2.

```
                  Host A                                Host B
          ------------------------                    ----------
          Address A1    Address A2                    Address B1
          ----------    ----------                    ----------
              |             |                             |
              |           SYN+MP_CAPABLE+L_RTT_EN         |
              |------------------------------------------>|
              |         SYN/ACK+MP_CAPABLE+L_RTT_EN       |
              |<------------------------------------------|
              |               ACK+MP_CAPABLE              |
              |------------------------------------------>|
              |      ACK+DSS+L_RTT_EN(latest RTT)+Data    |
              |<------------------------------------------|
              |             |       SYN+MP_JOIN           |
              |             |---------------------------->|
              |             |     MP_JOIN Process...      |
              |             |                             |
```

Figure 18: Negotiation procedure for RTT exchange

A successful negotiation allows the exchange of the measured RTT value from one subflow of a MPTCP host to another using the "Latest RTT" field within the L_RTT_EN option.

3.2.2. Protocol Adaptation

Calculating the "Latest RTT" by a remote host in an asymmetry transmission scenario should be transferred from remote host to the client running RobE_IPS. So a new MPTCP subtype option named L_RTT_EN is allocated for this function. During the three-way handshake L_RTT_EN is used for negotiation of remote RTT measurement capability between client and server (in Section 3.2.1). When both parts support the usage of remote RTT measurement, the "Latest RTT" field in L_RTT_EN is applied for carrying the value of latest RTT computed by the remote host.

```
                        1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +---------------+---------------+-------+-------+---------------+
     |     Kind      |    Length     |Subtype|     (reserved)        |
     +---------------+---------------+-------+-------+---------------+
     |                    Latest RTT (32 bits)                       |
     |                                                               |
     |                                                               |
     +---------------------------------------------------------------+
```

Figure 19: ROBE_L_RTT_EN OPTION

3.2.3. Fallback Mechanism

When the receiver is not L_RTT_EN capable, MPTCP session with L_RTT_EN negotiation will seamlessly fallback to normal MPTCP process.

[TBD, Need same checks as Section 3.1.4.2]

```
                       Host A                           Host B
          ------------------------                    ----------
          Address A1    Address A2                    Address B1
          ----------    ----------                    ----------
              |             |                             |
              |        SYN+MP_CAPABLE+L_RTT_EN            |
              |------------------------------------------>|
              |         SYN/ACK+MPTCP_CAPABLE             |
              |<------------------------------------------|
              |           ACK+MPTCP_CAPABLE               |
              |------------------------------------------>|
              |             |         SYN+MP_JOIN         |
              |             |---------------------------->|
              |             |       MP_JOIN Process...    |
              |             |                             |
```

Figure 20: Fallback to MPTCP without RobE_IPS

3.2.4. Security Consideration

[Tbd]

4. IANA Considerations

This document defines three new values to MPTCP Option Subtype as following.

Value	Symbol	Name	Reference
TBD	ROBE_eSIM_EN	RobE_eSIM enabled	Section 3.1
TBD	MP_JOIN_CAP	Join connection directly in RobE_eSIM	Section 3.1
TBD	L_RTT_EN	Server RTT enabled	Section 3.2

Table 2: RobE Option Subtypes

5. References

5.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.

[RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <https://www.rfc-editor.org/info/rfc8684>.

5.2. Informative References

[RobE_paper]
Amend, M., Rakocevic, V., Matz, A.P., and E. Bogenfeld, "RobE: Robust Connection Establishment for Multipath TCP", ANRW ’18 p. 76-82, 16 July 2018, <http://doi.acm.org/10.1145/3232755.3232762>.

[RobE_slides]
Amend, M., Matz, A.P., and E. Bogenfeld, "A proposal for MPTCP Robust session Establishment (MPTCP RobE)", IETF 99 Multipath TCP WG session, 18 July 2017, <https://datatracker.ietf.org/meeting/99/materials/slides-99-mptcp-a-proposal-for-mptcp-robust-session-establishment-mptcp-robe-01>.

Authors’ Addresses

Markus Amend
Deutsche Telekom
Deutsche-Telekom-Alle 9
64295 Darmstadt
Germany

Email: Markus.Amend@telekom.de

Jiao Kang
Huawei
D2-03,Huawei Industrial Base
Shenzhen
Guangdong, 518129
China

Email: kangjiao@huawei.com

TCPM Working Group C. Gomez
Internet-Draft UPC
Intended status: Experimental J. Crowcroft
Expires: May 4, 2021 University of Cambridge
October 31, 2020

TCP ACK Rate Request Option
draft-gomez-tcpm-ack-rate-request-01

Abstract

TCP Delayed Acknowledgments (ACKs) is a widely deployed mechanism that allows reducing protocol overhead in many scenarios. However, Delayed ACKs may also contribute to suboptimal performance. When a relatively large congestion window (cwnd) can be used, less frequent ACKs may be desirable. On the other hand, in relatively small cwnd scenarios, eliciting an immediate ACK may avoid unnecessary delays that may be incurred by the Delayed ACKs mechanism. This document specifies the TCP ACK Rate Request (TARR) option. This option allows a sender to indicate the ACK rate to be used by a receiver, and it also allows to request immediate ACKs from a receiver.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Delayed Acknowledgments (ACKs) were specified for TCP with the aim to reduce protocol overhead [RFC1122]. With Delayed ACKs, a TCP delays sending an ACK by up to 500 ms (often 200 ms, with lower values in recent implementations such as ~50 ms also reported), and typically sends an ACK for at least every second segment received in a stream of full-sized segments. This allows combining several segments into a single one (e.g. the application layer response to an application layer data message, and the corresponding ACK), and also saves up to one of every two ACKs, under many traffic patterns (e.g. bulk transfers). The "SHOULD" requirement level for implementing Delayed ACKs in RFC 1122, along with its expected benefits, has led to a widespread deployment of this mechanism.

However, there exist scenarios where Delayed ACKs contribute to suboptimal performance. We next roughly classify such scenarios into two main categories, in terms of the congestion window (cwnd) size and the Maximum Segment Size (MSS) that would be used therein: i) "large" cwnd scenarios (i.e. cwnd >> MSS), and ii) "small" cwnd scenarios (e.g. cwnd up to ~MSS).

In "large" cwnd scenarios, increasing the number of data segments after which a receiver transmits an ACK beyond the typical one (i.e. 2 when Delayed ACKs are used) may provide significant benefits. One example is mitigating performance limitations due to asymmetric path capacity (e.g. when the reverse path is significantly limited in comparison to the forward path) [RFC3449]. Another advantage is reducing the computational cost both at the sender and the receiver, and reducing network packet load, due to the lower number of ACKs involved.

In many "small" cwnd scenarios, a sender may want to request the receiver to acknowledge a data segment immediately (i.e. without the additional delay incurred by the Delayed ACKs mechanism). In high bit rate environments (e.g. data centers), a flow's fare share of the available Bandwidth Delay Product (BDP) may be in the order of one MSS, or even less. For an accordingly set cwnd value (e.g. cwnd up to MSS), Delayed ACKs would incur a delay that is several orders of magnitude greater than the RTT, severely degrading performance. Note that the Nagle algorithm may produce the same effect for some traffic patterns in the same type of environments [RFC8490]. In addition, when transactional data exchanges are performed over TCP, or when the cwnd size has been reduced, eliciting an immediate ACK from the receiver may avoid idle times and allow timely continuation of data transmission and/or cwnd growth, contributing to maintaining low latency.

Further "small" cwnd scenarios can be found in Internet of Things (IoT) environments. Many IoT devices exhibit significant memory constraints, such as only enough RAM for a send buffer size of 1 MSS. In that case, if the data segment does not elicit an application-layer response, the Delayed ACKs mechanism unnecessarily contributes a delay equal to the Delayed ACK timer to ACK transmission. The sender cannot transmit a new data segment until the ACK corresponding to the previous data segment is received and processed.

With the aim to provide a tool for performance improvement in both "large" and "small" cwnd scenarios, this document specifies the TCP ACK Rate request (TARR) option. This option allows a sender to indicate the ACK rate to be used by a receiver, and it also allows to request immediate ACKs from a receiver.

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 [RFC2119].

3. TCP ACK Rate Request Functionality

A TCP endpoint announces that it supports the TARR option by including the TARR option format in packets that have the SYN bit set. In such packets, the values carried by the TARR option format other than Kind, Length and ExID MUST be ignored by the receiving TCP.

TBD1: perhaps two formats (with one codepoint each), a shorter one just to advertise that TARR is supported, and a larger one which is the current TARR option format defined in section 4?

The next two subsections define the sender and receiver behaviors for devices that support the TARR option, respectively.

3.1. Sender behavior

A TCP sender MUST NOT include the TARR option in TCP data segments to be sent if the TCP receiver does not support the TARR option.

A TCP sender MAY request a TARR-option-capable receiver to modify the ACK rate of the latter to one ACK every R full-sized data segments received from the sender. This request is performed by the sender by including the TARR option in the TCP header of a data segment. The TARR option carries the R value requested by the sender (see section 4). For the described purpose, the value of R MUST NOT be zero. The TARR option also carries the N field, which MUST be ignored when R is not set to zero.

When a TCP sender needs a data segment to be acknowledged immediately by a TARR-option-capable receiving TCP, the sender includes the TARR option in the TCP header of the data segment, with a value of R equal to zero. When R is set to zero, the N field of the same option indicates the number of subsequent data segments for which the sender also requests immediate ACKs.

A TCP sender MAY indicate that it has a reordering tolerance of R packets by setting the Ignore Order field of the TARR option to True (see Section 4).

3.2. Receiver behavior

A receiving TCP conforming to this specification MUST process a TARR option present in a received data segment.

When the TARR option of a received segment carries an R value different from zero, a TARR-option-capable receiving TCP MUST modify its ACK rate to one ACK every R full-sized received data segments from the sender, as long as packet reordering does not occur. When R is different from zero, the receiving TCP MUST ignore the N field of the TARR option.

A TARR-option-capable TCP that receives a TARR option with the Ignore Order field set to True (see Section 4), MUST NOT send an ACK after each reordered data segment. Instead, it MUST continue to send one ACK every R received data segments. Otherwise (i.e., Ignore Order = False), such a receiver will need to send an ACK after each reordered data segment received.

If a TARR-option-capable TCP receives a segment carrying the TARR option with R=0, the receiving TCP MUST send an ACK immediately, and it MUST also send an ACK immediately after each one of the N next consecutive segments to be received. N refers to the corresponding field in the TARR option of the received segment (see Section 4).

4. Option Format

The TARR option has the format and content shown in Fig. 1.

```
          0        1        2        3
          01234567 89012345 67890123 45678901
         +--------+--------+--------+--------+
         |  Kind  | Length |       ExID      |
         +--------+--------+--------+--------+
         |   R    | IgnOrd |   N    |
         +--------+--------+--------+
```

Figure 1: TCP ACK Rate Request option format.

Kind: The Kind field value is TBD.

Length: The Length field value is 7 bytes.

ExID: The experiment ID field size is 2 bytes, and its value is 0x00AC.

R: The size of this field is 1 byte. If all bits of this field are set to 0, the field indicates a request by the sender for the receiver to trigger one or more ACKs immediately. Otherwise, the field carries the binary encoding of the number of full-sized segments received after which the receiver is requested by the sender to send an ACK.

Ignore Order: The size of this field is 1 byte. This field MUST have the value 0x01 ("True") or 0x00 ("False"). When this field is set to True, the receiver MUST NOT send an ACK after each reordered data segment. Instead, it MUST continue to send one ACK every R received data segments.

TBD2: perhaps 7 bits for R and 1 bit for Ignore Order?

N: The size of this field is 1 byte. When R=0, the N field indicates the number of subsequent consecutive data segments to be sent for which immediate ACKs are requested by the sender.

5. IANA Considerations

This document specifies a new TCP option (TCP ACK Rate Request) that uses the shared experimental options format [RFC6994], with ExID in network-standard byte order.

The authors plan to request the allocation of ExID value 0x00AC for the TCP option specified in this document.

6. Security Considerations

TBD

7. Acknowledgments

Bob Briscoe, Jonathan Morton, Richard Scheffenegger, Neal Cardwell, Michael Tuexen, Yuchung Cheng, Matt Mathis, Jana Iyengar, Gorry Fairhurst, and Stuart Cheshire provided useful comments and input for this document. Jana Iyengar suggested including a field to allow a sender communicate its tolerance to reordering. Gorry Fairhurst suggested adding a mechanism to request a number of consecutive immediate ACKs.

Carles Gomez has been funded in part by the Spanish Government through project PID2019-106808RA-I00, and by Secretaria d'Universitats i Recerca del Departament d'Empresa i Coneixement de la Generalitat de Catalunya 2017 through grant SGR 376.

8. References

8.1. Normative References

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <https://www.rfc-editor.org/info/rfc6994>.

8.2. Informative References

[RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <https://www.rfc-editor.org/info/rfc3449>.

[RFC8490] Bellis, R., Cheshire, S., Dickinson, J., Dickinson, S., Lemon, T., and T. Pusateri, "DNS Stateful Operations", RFC 8490, DOI 10.17487/RFC8490, March 2019, <https://www.rfc-editor.org/info/rfc8490>.

Authors’ Addresses

Carles Gomez
UPC
C/Esteve Terradas, 7
Castelldefels 08860
Spain

Email: carlesgo@entel.upc.edu

Jon Crowcroft
University of Cambridge
JJ Thomson Avenue
Cambridge, CB3 0FD
United Kingdom

Email: jon.crowcroft@cl.cam.ac.uk

TCP Maintenance & Minor Extensions (tcpm)
Internet-Draft
Updates: 3168, 3449 (if approved)
Intended status: Standards Track
Expires: May 6, 2021

B. Briscoe
Independent
M. Kuehlewind
Ericsson
R. Scheffenegger
NetApp

November 2, 2020

More Accurate ECN Feedback in TCP
draft-ietf-tcpm-accurate-ecn-13

Abstract

Explicit Congestion Notification (ECN) is a mechanism where network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the end-points. Receivers with an ECN-capable transport protocol feed back this information to the sender. ECN is specified for TCP in such a way that only one feedback signal can be transmitted per Round-Trip Time (RTT). Recent new TCP mechanisms like Congestion Exposure (ConEx), Data Center TCP (DCTCP) or Low Latency Low Loss Scalable Throughput (L4S) need more accurate ECN feedback information whenever more than one marking is received in one RTT. This document specifies a scheme to provide more than one feedback signal per RTT in the TCP header. Given TCP header space is scarce, it allocates a reserved header bit, that was previously used for the ECN-Nonce which has now been declared historic. It also overloads the two existing ECN flags in the TCP header. The resulting extra space is exploited to feed back the IP-ECN field received during the 3-way handshake as well. Supplementary feedback information can optionally be provided in a new TCP option, which is never used on the TCP SYN.

Status of This Memo

This Internet-Draft will expire on May 6, 2021.

Copyright Notice

Table of Contents

1. Introduction

Explicit Congestion Notification (ECN) [RFC3168] is a mechanism where network nodes can mark IP packets instead of dropping them to indicate incipient congestion to the end-points. Receivers with an ECN-capable transport protocol feed back this information to the sender. In RFC 3168, ECN was specified for TCP in such a way that only one feedback signal could be transmitted per Round-Trip Time (RTT). Recently, proposed mechanisms like Congestion Exposure (ConEx [RFC7713]), DCTCP [RFC8257] or L4S [I-D.ietf-tsvwg-l4s-arch] need to know when more than one marking is received in one RTT which is information that cannot be provided by the feedback scheme as specified in [RFC3168]. This document specifies an update to the ECN feedback scheme of RFC 3168 that provides more accurate information and could be used by these and potentially other future TCP extensions. A fuller treatment of the motivation for this specification is given in the associated requirements document [RFC7560].

This documents specifies a standards track scheme for ECN feedback in the TCP header to provide more than one feedback signal per RTT. It will be called the more accurate ECN feedback scheme, or AccECN for short. This document updates RFC 3168 with respect to negotiation and use of the feedback scheme for TCP. All aspects of RFC 3168 other than the TCP feedback scheme, in particular the definition of ECN at the IP layer, remain unchanged by this specification. Section 4 gives a more detailed specification of exactly which aspects of RFC 3168 this document updates.

AccECN is intended to be a complete replacement for classic TCP/ECN feedback, not a fork in the design of TCP. AccECN feedback complements TCP's loss feedback and it can coexist alongside 'classic' [RFC3168] TCP/ECN feedback. So its applicability is intended to include all public and private IP networks (and even any non-IP networks over which TCP is used today), whether or not any nodes on the path support ECN, of whatever flavour. This document uses the term Classic ECN when it needs to distinguish the RFC 3168 ECN TCP feedback scheme from the AccECN TCP feedback scheme.

AccECN feedback overloads the two existing ECN flags in the TCP header and allocates the currently reserved flag (previously called NS) in the TCP header, to be used as one three-bit counter field indicating the number of congestion experienced marked packets. Given the new definitions of these three bits, both ends have to support the new wire protocol before it can be used. Therefore during the TCP handshake the two ends use these three bits in the TCP header to negotiate the most advanced feedback protocol that they can both support, in a way that is backward compatible with [RFC3168].

AccECN is solely a change to the TCP wire protocol; it covers the negotiation and signaling of more accurate ECN feedback from a TCP Data Receiver to a Data Sender. It is completely independent of how TCP might respond to congestion feedback, which is out of scope, but ultimately the motivation for accurate ECN feedback. Like Classic ECN feedback, AccECN can be used by standard Reno congestion control [RFC5681] to respond to the existence of at least one congestion notification within a round trip. Or, unlike Reno, AccECN can be used to respond to the extent of congestion notification over a round trip, as for example DCTCP does in controlled environments [RFC8257]. For congestion response, this specification refers to RFC 3168, or ECN experiments such as those referred to in [RFC8311], namely: a TCP-based Low Latency Low Loss Scalable (L4S) congestion control [I-D.ietf-tsvwg-l4s-arch]; or Alternative Backoff with ECN (ABE) [RFC8511].

It is recommended that the AccECN protocol is implemented alongside SACK [RFC2018] and the experimental ECN++ protocol

[I-D.ietf-tcpm-generalized-ecn], which allows the ECN capability to be used on TCP control packets. Therefore, this specification does not discuss implementing AccECN alongside [RFC5562], which was an earlier experimental protocol with narrower scope than ECN++.

1.1. Document Roadmap

The following introductory section outlines the goals of AccECN (Section 1.2). Then terminology is defined (Section 1.3) and a recap of existing prerequisite technology is given (Section 1.4).

Section 2 gives an informative overview of the AccECN protocol. Then Section 3 gives the normative protocol specification, and Section 4 clarifies which aspects of RFC 3168 are updated by this specification. Section 5 assesses the interaction of AccECN with commonly used variants of TCP, whether standardized or not. Section 6 summarizes the features and properties of AccECN.

Section 7 summarizes the protocol fields and numbers that IANA will need to assign and Section 8 points to the aspects of the protocol that will be of interest to the security community.

Appendix A gives pseudocode examples for the various algorithms that AccECN uses and Appendix B explains why AccECN uses flags in the main TCP header and quantifies the space left for future use.

1.2. Goals

[RFC7560] enumerates requirements that a candidate feedback scheme will need to satisfy, under the headings: resilience, timeliness, integrity, accuracy (including ordering and lack of bias), complexity, overhead and compatibility (both backward and forward). It recognizes that a perfect scheme that fully satisfies all the requirements is unlikely and trade-offs between requirements are likely. Section 6 presents the properties of AccECN against these requirements and discusses the trade-offs made.

The requirements document recognizes that a protocol as ubiquitous as TCP needs to be able to serve as-yet-unspecified requirements. Therefore an AccECN receiver aims to act as a generic (dumb) reflector of congestion information so that in future new sender behaviours can be deployed unilaterally.

1.3. Terminology

AccECN: The more accurate ECN feedback scheme will be called AccECN for short.

Classic ECN: the ECN protocol specified in [RFC3168].

Classic ECN feedback: the feedback aspect of the ECN protocol specified in [RFC3168], including generation, encoding, transmission and decoding of feedback, but not the Data Sender's subsequent response to that feedback.

ACK: A TCP acknowledgement, with or without a data payload (ACK=1).

Pure ACK: A TCP acknowledgement without a data payload.

Acceptable packet / segment: A packet or segment that passes the acceptability tests in [RFC0793] and [RFC5961].

TCP client: The TCP stack that originates a connection.

TCP server: The TCP stack that responds to a connection request.

Data Receiver: The endpoint of a TCP half-connection that receives data and sends AccECN feedback.

Data Sender: The endpoint of a TCP half-connection that sends data and receives AccECN feedback.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.4. Recap of Existing ECN feedback in IP/TCP

ECN [RFC3168] uses two bits in the IP header. Once ECN has been negotiated with the receiver at the transport layer, an ECN sender can set two possible codepoints (ECT(0) or ECT(1)) in the IP header to indicate an ECN-capable transport (ECT). If both ECN bits are zero, the packet is considered to have been sent by a Not-ECN-capable Transport (Not-ECT). When a network node experiences congestion, it will occasionally either drop or mark a packet, with the choice depending on the packet's ECN codepoint. If the codepoint is Not-ECT, only drop is appropriate. If the codepoint is ECT(0) or ECT(1), the node can mark the packet by setting both ECN bits, which is termed 'Congestion Experienced' (CE), or loosely a 'congestion mark'. Table 1 summarises these codepoints.

IP-ECN codepoint	Codepoint name	Description
0b00	Not-ECT	Not ECN-Capable Transport
0b01	ECT(1)	ECN-Capable Transport (1)
0b10	ECT(0)	ECN-Capable Transport (0)
0b11	CE	Congestion Experienced

Table 1: The ECN Field in the IP Header

In the TCP header the first two bits in byte 14 are defined as flags for the use of ECN (CWR and ECE in Figure 1 [RFC3168]). A TCP client indicates it supports ECN by setting ECE=CWR=1 in the SYN, and an ECN-enabled server confirms ECN support by setting ECE=1 and CWR=0 in the SYN/ACK. On reception of a CE-marked packet at the IP layer, the Data Receiver starts to set the Echo Congestion Experienced (ECE) flag continuously in the TCP header of ACKs, which ensures the signal is received reliably even if ACKs are lost. The TCP sender confirms that it has received at least one ECE signal by responding with the congestion window reduced (CWR) flag, which allows the TCP receiver to stop repeating the ECN-Echo flag. This always leads to a full RTT of ACKs with ECE set. Thus any additional CE markings arriving within this RTT cannot be fed back.

The last bit in byte 13 of the TCP header was defined as the Nonce Sum (NS) for the ECN Nonce [RFC3540]. In the absence of widespread deployment RFC 3540 has been reclassified as historic [RFC8311] and the respective flag has been marked as "reserved", making this TCP flag available for use by the AccECN experiment instead.

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               |           | N | C | E | U | A | P | R | S | F |
| Header Length | Reserved  | S | W | C | R | C | S | S | Y | I |
|               |           |   | R | E | G | K | H | T | N | N |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

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

2. AccECN Protocol Overview and Rationale

This section provides an informative overview of the AccECN protocol that will be normatively specified in Section 3

Like the original TCP approach, the Data Receiver of each TCP half-connection sends AccECN feedback to the Data Sender on TCP acknowledgements, reusing data packets of the other half-connection whenever possible.

The AccECN protocol has had to be designed in two parts:

- an essential part that re-uses ECN TCP header bits to feed back the number of arriving CE marked packets. This provides more accuracy than classic ECN feedback, but limited resilience against ACK loss;

- a supplementary part using a new AccECN TCP Option that provides additional feedback on the number of bytes that arrive marked with each of the three ECN codepoints (not just CE marks). This provides greater resilience against ACK loss than the essential feedback, but it is more likely to suffer from middlebox interference.

The two part design was necessary, given limitations on the space available for TCP options and given the possibility that certain incorrectly designed middleboxes prevent TCP using any new options.

The essential part overloads the previous definition of the three flags in the TCP header that had been assigned for use by ECN. This design choice deliberately replaces the classic ECN feedback protocol, rather than leaving classic ECN feedback intact and adding more accurate feedback separately because:

- this efficiently reuses scarce TCP header space, given TCP option space is approaching saturation;

- a single upgrade path for the TCP protocol is preferable to a fork in the design;

- otherwise classic and accurate ECN feedback could give conflicting feedback on the same segment, which could open up new security concerns and make implementations unnecessarily complex;

- middleboxes are more likely to faithfully forward the TCP ECN flags than newly defined areas of the TCP header.

AccECN is designed to work even if the supplementary part is removed or zeroed out, as long as the essential part gets through.

2.1. Capability Negotiation

AccECN is a change to the wire protocol of the main TCP header, therefore it can only be used if both endpoints have been upgraded to understand it. The TCP client signals support for AccECN on the initial SYN of a connection and the TCP server signals whether it supports AccECN on the SYN/ACK. The TCP flags on the SYN that the client uses to signal AccECN support have been carefully chosen so that a TCP server will interpret them as a request to support the most recent variant of ECN feedback that it supports. Then the client falls back to the same variant of ECN feedback.

An AccECN TCP client does not send the new AccECN Option on the SYN as SYN option space is limited. The TCP server sends the AccECN Option on the SYN/ACK and the client sends it on the first ACK to test whether the network path forwards the option correctly.

2.2. Feedback Mechanism

A Data Receiver maintains four counters initialized at the start of the half-connection. Three count the number of arriving payload bytes marked CE, ECT(1) and ECT(0) respectively. The fourth counts the number of packets arriving marked with a CE codepoint (including control packets without payload if they are CE-marked).

The Data Sender maintains four equivalent counters for the half connection, and the AccECN protocol is designed to ensure they will match the values in the Data Receiver's counters, albeit after a little delay.

Each ACK carries the three least significant bits (LSBs) of the packet-based CE counter using the ECN bits in the TCP header, now renamed the Accurate ECN (ACE) field (see Figure 3 later). The 24 LSBs of each byte counter are carried in the AccECN Option.

2.3. Delayed ACKs and Resilience Against ACK Loss

With both the ACE and the AccECN Option mechanisms, the Data Receiver continually repeats the current LSBs of each of its respective counters. There is no need to acknowledge these continually repeated counters, so the congestion window reduced (CWR) mechanism is no longer used. Even if some ACKs are lost, the Data Sender should be able to infer how much to increment its own counters, even if the protocol field has wrapped.

The 3-bit ACE field can wrap fairly frequently. Therefore, even if it appears to have incremented by one (say), the field might have actually cycled completely then incremented by one. The Data Receiver is not allowed to delay sending an ACK to such an extent that the ACE field would cycle. However cycling is still a possibility at the Data Sender because a whole sequence of ACKs carrying intervening values of the field might all be lost or delayed in transit.

The fields in the AccECN Option are larger, but they will increment in larger steps because they count bytes not packets. Nonetheless, their size has been chosen such that a whole cycle of the field would never occur between ACKs unless there had been an infeasibly long sequence of ACK losses. Therefore, as long as the AccECN Option is available, it can be treated as a dependable feedback channel.

If the AccECN Option is not available, e.g. it is being stripped by a middlebox, the AccECN protocol will only feed back information on CE markings (using the ACE field). Although not ideal, this will be sufficient, because it is envisaged that neither ECT(0) nor ECT(1) will ever indicate more severe congestion than CE, even though future uses for ECT(0) or ECT(1) are still unclear [RFC8311]. Because the 3-bit ACE field is so small, when it is the only field available the Data Sender has to interpret it assuming the most likely wrap, but with a degree of conservatism.

Certain specified events trigger the Data Receiver to include an AccECN Option on an ACK. The rules are designed to ensure that the order in which different markings arrive at the receiver is communicated to the sender (as long as options are reaching the sender and as long as there is no ACK loss). Implementations are encouraged to send an AccECN Option more frequently, but this is left up to the implementer.

2.4. Feedback Metrics

The CE packet counter in the ACE field and the CE byte counter in the AccECN Option both provide feedback on received CE-marks. The CE packet counter includes control packets that do not have payload data, while the CE byte counter solely includes marked payload bytes. If both are present, the byte counter in the option will provide the more accurate information needed for modern congestion control and policing schemes, such as L4S, DCTCP or ConEx. If the option is stripped, a simple algorithm to estimate the number of marked bytes from the ACE field is given in Appendix A.3.

Feedback in bytes is recommended in order to protect against the receiver using attacks similar to 'ACK-Division' to artificially inflate the congestion window, which is why [RFC5681] now recommends that TCP counts acknowledged bytes not packets.

2.5. Generic (Dumb) Reflector

The ACE field provides information about CE markings on both data and control packets. According to [RFC3168] the Data Sender is meant to set control packets to Not-ECT. However, mechanisms in certain private networks (e.g. data centres) set control packets to be ECN capable because they are precisely the packets that performance depends on most.

For this reason, AccECN is designed to be a generic reflector of whatever ECN markings it sees, whether or not they are compliant with a current standard. Then as standards evolve, Data Senders can upgrade unilaterally without any need for receivers to upgrade too. It is also useful to be able to rely on generic reflection behaviour when senders need to test for unexpected interference with markings (for instance Section 3.2.2.3, Section 3.2.2.4 and Section 3.2.3.2 of the present document and para 2 of Section 20.2 of [RFC3168]).

The initial SYN is the most critical control packet, so AccECN provides feedback on its ECN marking. Although RFC 3168 prohibits an ECN-capable SYN, providing feedback of ECN marking on the SYN supports future scenarios in which SYNs might be ECN-enabled (without prejudging whether they ought to be). For instance, [RFC8311] updates this aspect of RFC 3168 to allow experimentation with ECN-capable TCP control packets.

Even if the TCP client (or server) has set the SYN (or SYN/ACK) to not-ECT in compliance with RFC 3168, feedback on the state of the ECN field when it arrives at the receiver could still be useful, because middleboxes have been known to overwrite the ECN IP field as if it is still part of the old Type of Service (ToS) field [Mandalari18]. If a TCP client has set the SYN to Not-ECT, but receives feedback that the ECN field on the SYN arrived with a different codepoint, it can detect such middlebox interference and send Not-ECT for the rest of the connection. Today, if a TCP server receives ECT or CE on a SYN, it cannot know whether it is invalid (or valid) because only the TCP client knows whether it originally marked the SYN as Not-ECT (or ECT). Therefore, prior to AccECN, the server's only safe course of action was to disable ECN for the connection. Instead, the AccECN protocol allows the server to feed back the received ECN field to the client, which then has all the information to decide whether the connection has to fall-back from supporting ECN (or not).

3. AccECN Protocol Specification

3.1. Negotiating to use AccECN

3.1.1. Negotiation during the TCP handshake

Given the ECN Nonce [RFC3540] has been reclassified as historic [RFC8311], the present specification re-allocates the TCP flag at bit 7 of the TCP header, which was previously called NS (Nonce Sum), as the AE (Accurate ECN) flag (see IANA Considerations in Section 7) as shown below.

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               |           | A | C | E | U | A | P | R | S | F |
| Header Length | Reserved  | E | W | C | R | C | S | S | Y | I |
|               |           |   | R | E | G | K | H | T | N | N |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Figure 2: The (post-AccECN) definition of the TCP header flags during the TCP handshake

During the TCP handshake at the start of a connection, to request more accurate ECN feedback the TCP client (host A) MUST set the TCP flags AE=1, CWR=1 and ECE=1 in the initial SYN segment.

If a TCP server (B) that is AccECN-enabled receives a SYN with the above three flags set, it MUST set both its half connections into AccECN mode. Then it MUST set the TCP flags on the SYN/ACK to one of the 4 values shown in the top block of Table 2 to confirm that it supports AccECN. The TCP server MUST NOT set one of these 4 combination of flags on the SYN/ACK unless the preceding SYN requested support for AccECN as above.

A TCP server in AccECN mode MUST set the AE, CWR and ECE TCP flags on the SYN/ACK to the value in Table 2 that feeds back the IP-ECN field that arrived on the SYN. This applies whether or not the server itself supports setting the IP-ECN field on a SYN or SYN/ACK (see Section 2.5 for rationale).

Once a TCP client (A) has sent the above SYN to declare that it supports AccECN, and once it has received the above SYN/ACK segment that confirms that the TCP server supports AccECN, the TCP client MUST set both its half connections into AccECN mode.

Once in AccECN mode, a TCP client or server has the rights and obligations to participate in the ECN protocol defined in Section 3.1.5.

The procedure for the client to follow if a SYN/ACK does not arrive before its retransmission timer expires is given in Section 3.1.4.

3.1.2. Backward Compatibility

The three flags set to 1 to indicate AccECN support on the SYN have been carefully chosen to enable natural fall-back to prior stages in the evolution of ECN, as above. Table 2 tabulates all the negotiation possibilities for ECN-related capabilities that involve at least one AccECN-capable host. The entries in the first two columns have been abbreviated, as follows:

AccECN: More Accurate ECN Feedback (the present specification)

Nonce: ECN Nonce feedback [RFC3540]

ECN: 'Classic' ECN feedback [RFC3168]

No ECN: Not-ECN-capable. Implicit congestion notification using packet drop.

A	B	SYN A->B			SYN/ACK B->A			Feedback Mode
		AE	CWR	ECE	AE	CWR	ECE	
AccECN	AccECN	1	1	1	0	1	0	AccECN (no ECT on SYN)
AccECN	AccECN	1	1	1	0	1	1	AccECN (ECT1 on SYN)
AccECN	AccECN	1	1	1	1	0	0	AccECN (ECT0 on SYN)
AccECN	AccECN	1	1	1	1	1	0	AccECN (CE on SYN)
AccECN	Nonce	1	1	1	1	0	1	(Reserved)
AccECN	ECN	1	1	1	0	0	1	classic ECN
AccECN	No ECN	1	1	1	0	0	0	Not ECN
Nonce	AccECN	0	1	1	0	0	1	classic ECN
ECN	AccECN	0	1	1	0	0	1	classic ECN
No ECN	AccECN	0	0	0	0	0	0	Not ECN
AccECN	Broken	1	1	1	1	1	1	Not ECN

Table 2: ECN capability negotiation between Client (A) and Server (B)

Table 2 is divided into blocks each separated by an empty row.

1. The top block shows the case already described in Section 3.1 where both endpoints support AccECN and how the TCP server (B) indicates congestion feedback.

2. The second block shows the cases where the TCP client (A) supports AccECN but the TCP server (B) supports some earlier variant of TCP feedback, indicated in its SYN/ACK. Therefore, as soon as an AccECN-capable TCP client (A) receives the SYN/ACK shown it MUST set both its half connections into the feedback mode shown in the rightmost column. If it has set itself into classic ECN feedback mode it MUST then comply with [RFC3168].

The server response called 'Nonce' in the table is now historic. For an AccECN implementation, there is no need to recognize or support ECN Nonce feedback [RFC3540], which has been reclassified as historic [RFC8311]. AccECN is compatible with alternative ECN feedback integrity approaches (see Section 5.3).

3. The third block shows the cases where the TCP server (B) supports AccECN but the TCP client (A) supports some earlier variant of TCP feedback, indicated in its SYN.

   When an AccECN-enabled TCP server (B) receives a SYN with AE,CWR,ECE = 0,1,1 it MUST do one of the following:

   * set both its half connections into the classic ECN feedback mode and return a SYN/ACK with AE, CWR, ECE = 0,0,1 as shown. Then it MUST comply with [RFC3168].

   * set both its half-connections into No ECN mode and return a SYN/ACK with AE,CWR,ECE = 0,0,0, then continue with ECN disabled. This latter case is unlikely to be desirable, but it is allowed as a possibility, e.g. for minimal TCP implementations.

   When an AccECN-enabled TCP server (B) receives a SYN with AE,CWR,ECE = 0,0,0 it MUST set both its half connections into the Not ECN feedback mode, return a SYN/ACK with AE,CWR,ECE = 0,0,0 as shown and continue with ECN disabled.

4. The fourth block displays a combination labelled 'Broken'. Some older TCP server implementations incorrectly set the reserved flags in the SYN/ACK by reflecting those in the SYN. Such broken TCP servers (B) cannot support ECN, so as soon as an AccECN-capable TCP client (A) receives such a broken SYN/ACK it MUST fall back to Not ECN mode for both its half connections and continue with ECN disabled.

The following additional rules do not fit the structure of the table, but they complement it:

Simultaneous Open: An originating AccECN Host (A), having sent a SYN with AE=1, CWR=1 and ECE=1, might receive another SYN from host B. Host A MUST then enter the same feedback mode as it would have entered had it been a responding host and received the same SYN. Then host A MUST send the same SYN/ACK as it would have sent had it been a responding host.

In-window SYN during TIME-WAIT: Many TCP implementations create a new TCP connection if they receive an in-window SYN packet during

TIME-WAIT state. When a TCP host enters TIME-WAIT or CLOSED state, it should ignore any previous state about the negotiation of AccECN for that connection and renegotiate the feedback mode according to Table 2.

3.1.3. Forward Compatibility

If a TCP server that implements AccECN receives a SYN with the three TCP header flags (AE, CWR and ECE) set to any combination other than 000, 011 or 111, it MUST negotiate the use of AccECN as if they had been set to 111. This ensures that future uses of the other combinations on a SYN can rely on consistent behaviour from the installed base of AccECN servers.

For the avoidance of doubt, the behaviour described in the present specification applies whether or not the three remaining reserved TCP header flags are zero.

3.1.4. Retransmission of the SYN

If the sender of an AccECN SYN times out before receiving the SYN/ACK, the sender SHOULD attempt to negotiate the use of AccECN at least one more time by continuing to set all three TCP ECN flags on the first retransmitted SYN (using the usual retransmission time-outs). If this first retransmission also fails to be acknowledged, the sender SHOULD send subsequent retransmissions of the SYN with the three TCP-ECN flags cleared (AE=CWR=ECE=0). A retransmitted SYN MUST use the same ISN as the original SYN.

Retrying once before fall-back adds delay in the case where a middlebox drops an AccECN (or ECN) SYN deliberately. However, current measurements imply that a drop is less likely to be due to middlebox interference than other intermittent causes of loss, e.g. congestion, wireless interference, etc.

Implementers MAY use other fall-back strategies if they are found to be more effective (e.g. attempting to negotiate AccECN on the SYN only once or more than twice (most appropriate during high levels of congestion). However, other fall-back strategies will need to follow all the rules in Section 3.1.5, which concern behaviour when SYNs or SYN/ACKs negotiating different types of feedback have been sent within the same connection.

Further it may make sense to also remove any other new or experimental fields or options on the SYN in case a middlebox might be blocking them, although the required behaviour will depend on the specification of the other option(s) and any attempt to co-ordinate fall-back between different modules of the stack.

Whichever fall-back strategy is used, the TCP initiator SHOULD cache failed connection attempts. If it does, it SHOULD NOT give up attempting to negotiate AccECN on the SYN of subsequent connection attempts until it is clear that the blockage is persistently and specifically due to AccECN. The cache should be arranged to expire so that the initiator will infrequently attempt to check whether the problem has been resolved.

The fall-back procedure if the TCP server receives no ACK to acknowledge a SYN/ACK that tried to negotiate AccECN is specified in Section 3.2.3.2.

3.1.5. Implications of AccECN Mode

Section 3.1.1 describes the only ways that a host can enter AccECN mode, whether as a client or as a server.

As a Data Sender, a host in AccECN mode has the rights and obligations concerning the use of ECN defined below, which build on those in [RFC3168] as updated by [RFC8311]:

- Using ECT:
    - It can set an ECT codepoint in the IP header of packets to indicate to the network that the transport is capable and willing to participate in ECN for this packet.
    - It does not have to set ECT on any packet (for instance if it has reason to believe such a packet would be blocked).
- Switching feedback negotiation (e.g. fall-back):
    - It SHOULD NOT set ECT on any packet if it has received at least one valid SYN or Acceptable SYN/ACK with AE=CWR=ECE=0. A "valid SYN" has the same port numbers and the same ISN as the SYN that caused the server to enter AccECN mode.
    - It MUST NOT send an ECN-setup SYN [RFC3168] within the same connection as it has sent a SYN requesting AccECN feedback.
    - It MUST NOT send an ECN-setup SYN/ACK [RFC3168] within the same connection as it has sent a SYN/ACK agreeing to use AccECN feedback.

  The above rules are necessary because, when one peer negotiates the feedback mode in two different types of handshake, it is not possible for the other peer to know for certain which handshake packet(s) the other end eventually receives or in which order it receives them. So the two peers can end up using difference feedback modes without knowing it.

- Congestion response:

  * It is still obliged to respond appropriately to AccECN feedback with congestion indications on packets it had previously sent, as defined in Section 6.1 of [RFC3168] and updated by Sections 2.1 and 4.1 of [RFC8311].

  * The commitment to respond appropriately to incoming indications of congestion remains even if it sends a SYN packet with AE=CWR=ECE=0, in a later transmission within the same TCP connection.

  * Unlike an RFC 3168 data sender, it MUST NOT set CWR to indicate it has received and responded to indications of congestion (for the avoidance of doubt, this does not preclude it from setting the bits of the ACE counter field, which includes an overloaded use of the same bit).

As a Data Receiver:

- a host in AccECN mode MUST feed back the information in the IP-ECN field on incoming packets using Accurate ECN feedback, as specified in Section 3.2 below.

- if it receives an ECN-setup SYN or ECN-setup SYN/ACK [RFC3168] during the same connection as it receives a SYN requesting AccECN feedback or a SYN/ACK agreeing to use AccECN feedback, it MUST reset the connection with a RST packet.

- If for any reason it is not willing to provide ECN feedback on a particular TCP connection, to indicate this unwillingness it SHOULD clear the AE, CWR and ECE flags in all SYN and/or SYN/ACK packets that it sends.

- it MUST NOT use reception of packets with ECT set in the IP-ECN field as an implicit signal that the peer is ECN-capable. Reason: ECT at the IP layer does not explicitly confirm the peer has the correct ECN feedback logic, and the packets could have been mangled at the IP layer.

3.2. AccECN Feedback

Each Data Receiver of each half connection maintains four counters, r.cep, r.ceb, r.e0b and r.e1b:

- The Data Receiver MUST increment the CE packet counter (r.cep), for every Acceptable packet that it receives with the CE code point in the IP ECN field, including CE marked control packets but excluding CE on SYN packets (SYN=1; ACK=0).

- The Data Receiver MUST increment the r.ceb, r.e0b or r.e1b byte counters by the number of TCP payload octets in Acceptable packets marked respectively with the CE, ECT(0) and ECT(1) codepoint in their IP-ECN field, including any payload octets on control packets, but not including any payload octets on SYN packets (SYN=1; ACK=0).

Each Data Sender of each half connection maintains four counters, s.cep, s.ceb, s.e0b and s.e1b intended to track the equivalent counters at the Data Receiver.

A Data Receiver feeds back the CE packet counter using the Accurate ECN (ACE) field, as explained in Section 3.2.2. And it feeds back all the byte counters using the AccECN TCP Option, as specified in Section 3.2.3.

Whenever a host feeds back the value of any counter, it MUST report the most recent value, no matter whether it is in a pure ACK, an ACK with new payload data or a retransmission. Therefore the feedback carried on a retransmitted packet is unlikely to be the same as the feedback on the original packet.

3.2.1. Initialization of Feedback Counters

When a host first enters AccECN mode, in its role as a Data Receiver it initializes its counters to r.cep = 5, r.e0b = 1 and r.ceb = r.e1b.= 0,

Non-zero initial values are used to support a stateless handshake (see Section 5.1) and to be distinct from cases where the fields are incorrectly zeroed (e.g. by middleboxes - see Section 3.2.3.2.4).

When a host enters AccECN mode, in its role as a Data Sender it initializes its counters to s.cep = 5, s.e0b = 1 and s.ceb = s.e1b.= 0.

3.2.2. The ACE Field

After AccECN has been negotiated on the SYN and SYN/ACK, both hosts overload the three TCP flags (AE, CWR and ECE) in the main TCP header as one 3-bit field. Then the field is given a new name, ACE, as shown in Figure 3.

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               |           |           | U | A | P | R | S | F |
| Header Length | Reserved  |    ACE    | R | C | S | S | Y | I |
|               |           |           | G | K | H | T | N | N |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Figure 3: Definition of the ACE field within bytes 13 and 14 of the TCP Header (when AccECN has been negotiated and SYN=0).

The original definition of these three flags in the TCP header, including the addition of support for the ECN Nonce, is shown for comparison in Figure 1. This specification does not rename these three TCP flags to ACE unconditionally; it merely overloads them with another name and definition once an AccECN connection has been established.

With one exception (Section 3.2.2.1), a host with both of its half-connections in AccECN mode MUST interpret the AE, CWR and ECE flags as the 3-bit ACE counter on a segment with the SYN flag cleared (SYN=0). On such a packet, a Data Receiver MUST encode the three least significant bits of its r.cep counter into the ACE field that it feeds back to the Data Sender. A host MUST NOT interpret the 3 flags as a 3-bit ACE field on any segment with SYN=1 (whether ACK is 0 or 1), or if AccECN negotiation is incomplete or has not succeeded.

Both parts of each of these conditions are equally important. For instance, even if AccECN negotiation has been successful, the ACE field is not defined on any segments with SYN=1 (e.g. a retransmission of an unacknowledged SYN/ACK, or when both ends send SYN/ACKs after AccECN support has been successfully negotiated during a simultaneous open).

3.2.2.1. ACE Field on the ACK of the SYN/ACK

A TCP client (A) in AccECN mode MUST feed back which of the 4 possible values of the IP-ECN field was on the SYN/ACK by writing it into the ACE field of a pure ACK with no SACK blocks using the binary encoding in Table 3 (which is the same as that used on the SYN/ACK in Table 2). This shall be called the handshake encoding of the ACE field, and it is the only exception to the rule that the ACE field carries the 3 least significant bits of the r.cep counter on packets with SYN=0.

Normally, a TCP client acknowledges a SYN/ACK with an ACK that satisfies the above conditions anyway (SYN=0, no data, no SACK blocks). If an AccECN TCP client intends to acknowledge the SYN/ACK with a packet that does not satisfy these conditions (e.g. it has data to include on the ACK), it SHOULD first send a pure ACK that does satisfy these conditions (see Section 5.2), so that it can feed back which of the four values of the IP-ECN field arrived on the SYN/ACK. A valid exception to this "SHOULD" would be where the implementation will only be used in an environment where mangling of the ECN field is unlikely.

IP-ECN codepoint on SYN/ACK	ACE on pure ACK of SYN/ACK	r.cep of client in AccECN mode
Not-ECT	0b010	5
ECT(1)	0b011	5
ECT(0)	0b100	5
CE	0b110	6

Table 3: The encoding of the ACE field in the ACK of the SYN-ACK to reflect the SYN-ACK's IP-ECN field

When an AccECN server in SYN-RCVD state receives a pure ACK with SYN=0 and no SACK blocks, instead of treating the ACE field as a counter, it MUST infer the meaning of each possible value of the ACE field from Table 4, which also shows the value that an AccECN server MUST set s.cep to as a result.

Given this encoding of the ACE field on the ACK of a SYN/ACK is exceptional, an AccECN server using large receive offload (LRO) might prefer to disable LRO until such an ACK has transitioned it out of SYN-RCVD state.

ACE on ACK of SYN/ACK	IP-ECN codepoint on SYN/ACK inferred by server	s.cep of server in AccECN mode
0b000	{Notes 1, 3}	Disable ECN
0b001	{Notes 2, 3}	5
0b010	Not-ECT	5
0b011	ECT(1)	5
0b100	ECT(0)	5
0b101	Currently Unused {Note 2}	5
0b110	CE	6
0b111	Currently Unused {Note 2}	5

Table 4: Meaning of the ACE field on the ACK of the SYN/ACK

{Note 1}: If the server is in AccECN mode, the value of zero raises suspicion of zeroing of the ACE field on the path (see Section 3.2.2.3).

{Note 2}: If the server is in AccECN mode, these values are Currently Unused but the AccECN server's behaviour is still defined for forward compatibility. Then the designer of a future protocol can know for certain what AccECN servers will do with these codepoints.

{Note 3}: In the case where a server that implements AccECN is also using a stateless handshake (termed a SYN cookie) it will not remember whether it entered AccECN mode. The values 0b000 or 0b001 will remind it that it did not enter AccECN mode, because AccECN does not use them (see Section 5.1 for details). If a stateless server that implements AccECN receives either of these two values in the ACK, its action is implementation-dependent and outside the scope of this spec, It will certainly not take the action in the third column because, after it receives either of these values, it is not in AccECN mode. I.e., it will not disable ECN (at least not just because ACE is 0b000) and it will not set s.cep.

3.2.2.2. Encoding and Decoding Feedback in the ACE Field

Whenever the Data Receiver sends an ACK with SYN=0 (with or without data), unless the handshake encoding in Section 3.2.2.1 applies, the Data Receiver MUST encode the least significant 3 bits of its r.cep counter into the ACE field (see Appendix A.2).

Whenever the Data Sender receives an ACK with SYN=0 (with or without data), it first checks whether it has already been superseded by another ACK in which case it ignores the ECN feedback. If the ACK has not been superseded, and if the special handshake encoding in Section 3.2.2.1 does not apply, the Data Sender decodes the ACE field as follows (see Appendix A.2 for examples).

o It takes the least significant 3 bits of its local s.cep counter and subtracts them from the incoming ACE counter to work out the minimum positive increment it could apply to s.cep (assuming the ACE field only wrapped at most once).

o It then follows the safety procedures in Section 3.2.2.5.2 to calculate or estimate how many packets the ACK could have acknowledged under the prevailing conditions to determine whether the ACE field might have wrapped more than once.

The encode/decode procedures during the three-way handshake are exceptions to the general rules given so far, so they are spelled out step by step below for clarity:

- If a TCP server in AccECN mode receives a CE mark in the IP-ECN field of a SYN (SYN=1, ACK=0), it MUST NOT increment r.cep (it remains at its initial value of 5).

  Reason: It would be redundant for the server to include CE-marked SYNs in its r.cep counter, because it already reliably delivers feedback of any CE marking on the SYN/ACK using the encoding in Table 2. This also ensures that, when the server starts using the ACE field, it has not unnecessarily consumed more than one initial value, given they can be used to negotiate variants of the AccECN protocol (see Appendix B.3).

- If a TCP client in AccECN mode receives CE feedback in the TCP flags of a SYN/ACK, it MUST NOT increment s.cep (it remains at its initial value of 5), so that it stays in step with r.cep on the server. Nonetheless, the TCP client still triggers the congestion control actions necessary to respond to the CE feedback.

- If a TCP client in AccECN mode receives a CE mark in the IP-ECN field of a SYN/ACK, it MUST increment r.cep, but no more than once no matter how many CE-marked SYN/ACKs it receives (i.e. incremented from 5 to 6, but no further).

  Reason: Incrementing r.cep ensures the client will eventually deliver any CE marking to the server reliably when it starts using the ACE field. Even though the client also feeds back any CE marking on the ACK of the SYN/ACK using the encoding in Table 3, this ACK is not delivered reliably, so it can be considered as a timely notification that is redundant but unreliable. The client does not increment r.cep more than once, because the server can only increment s.cep once (see next bullet). Also, this limits the unnecessarily consumed initial values of the ACE field to two.

- If a TCP server in AccECN mode and in SYN-RCVD state receives CE feedback in the TCP flags of a pure ACK with no SACK blocks, it MUST increment s.cep (from 5 to 6). The TCP server then triggers the congestion control actions necessary to respond to the CE feedback.

  Reasoning: The TCP server can only increment s.cep once, because the first ACK it receives will cause it to transition out of SYN-RCVD state. The server’s congestion response would be no different even if it could receive feedback of more than one CE-marked SYN/ACK.

  Once the TCP server transitions to ESTABLISHED state, it might later receive other pure ACK(s) with the handshake encoding in the ACE field. The conditions for this to occur are quite unusual, but not impossible, e.g. a SYN/ACK (or ACK of the SYN/ACK) that is delayed for longer than the server's retransmission timeout; or packet duplication by the network. Nonetheless, once in the ESTABLISHED state, the server will consider the ACE field to be encoded as the normal ACE counter on all packets with SYN=0 (given it will be following the above rule in this bullet). The server MAY include a test to avoid this case.

3.2.2.3. Testing for Zeroing of the ACE Field

Section 3.2.2 required the Data Receiver to initialize the r.cep counter to a non-zero value. Therefore, in either direction the initial value of the ACE counter ought to be non-zero.

If AccECN has been successfully negotiated, the Data Sender SHOULD check the value of the ACE counter in the first packet (with or without data) that arrives with SYN=0. If the value of this ACE field is zero (0b000), the Data Sender disables sending ECN-capable packets for the remainder of the half-connection by setting the IP/ECN field in all subsequent packets to Not-ECT.

Usually, the server checks the ACK of the SYN/ACK from the client, while the client checks the first data segment from the server. However, if reordering occurs, "the first packet ... that arrives" will not necessarily be the same as the first packet in sequence order. The test has been specified loosely like this to simplify implementation, and because it would not have been any more precise to have specified the first packet in sequence order, which would not necessarily be the first ACE counter that the Data Receiver fed back anyway, given it might have been a retransmission.

The possibility of re-ordering means that there is a small chance that the ACE field on the first packet to arrive is genuinely zero (without middlebox interference). This would cause a host to unnecessarily disable ECN for a half connection. Therefore, in environments where there is no evidence of the ACE field being zeroed, implementations can skip this test.

Note that the Data Sender MUST NOT test whether the arriving counter in the initial ACE field has been initialized to a specific valid value - the above check solely tests whether the ACE fields have been incorrectly zeroed. This allows hosts to use different initial values as an additional signalling channel in future.

3.2.2.4. Testing for Mangling of the IP/ECN Field

The value of the ACE field on the SYN/ACK indicates the value of the IP/ECN field when the SYN arrived at the server. The client can compare this with how it originally set the IP/ECN field on the SYN. If this comparison implies an unsafe transition (see below) of the IP/ECN field, for the remainder of the connection the client MUST NOT send ECN-capable packets, but it MUST continue to feed back any ECN markings on arriving packets.

The value of the ACE field on the last ACK of the 3WHS indicates the value of the IP/ECN field when the SYN/ACK arrived at the client. The server can compare this with how it originally set the IP/ECN field on the SYN/ACK. If this comparison implies an unsafe transition of the IP/ECN field, for the remainder of the connection the server MUST NOT send ECN-capable packets, but it MUST continue to feed back any ECN markings on arriving packets.

The ACK of the SYN/ACK is not reliably delivered (nonetheless, the count of CE marks is still eventually delivered reliably). If this ACK does not arrive, the server can continue to send ECN-capable packets without having tested for mangling of the IP/ECN field on the SYN/ACK.

Invalid transitions of the IP/ECN field are defined in [RFC3168] and repeated here for convenience:

- the not-ECT codepoint changes;

- either ECT codepoint transitions to not-ECT;

- the CE codepoint changes.

RFC 3168 says that a router that changes ECT to not-ECT is invalid but safe. However, from a host's viewpoint, this transition is unsafe because it could be the result of two transitions at different routers on the path: ECT to CE (safe) then CE to not-ECT (unsafe). This scenario could well happen where an ECN-enabled home router congests its upstream mobile broadband bottleneck link, then the ingress to the mobile network clears the ECN field [Mandalari18].

Once a Data Sender has entered AccECN mode it SHOULD check whether all feedback received for the first three or four round indicated that every packet it sent was CE-marked. If so, for the remainder of the connection, the Data Sender SHOULD NOT send ECN-capable packets, but it MUST continue to feed back any ECN markings on arriving packets.

The above fall-back behaviours are necessary in case mangling of the IP/ECN field is asymmetric, which is currently common over some mobile networks [Mandalari18]. Then one end might see no unsafe transition and continue sending ECN-capable packets, while the other end sees an unsafe transition and stops sending ECN-capable packets.

3.2.2.5. Safety against Ambiguity of the ACE Field

If too many CE-marked segments are acknowledged at once, or if a long run of ACKs is lost or thinned out, the 3-bit counter in the ACE field might have cycled between two ACKs arriving at the Data Sender. The following safety procedures minimize this ambiguity.

3.2.2.5.1. Data Receiver Safety Procedures

An AccECN Data Receiver:

- SHOULD immediately send an ACK whenever a data packet marked CE arrives after the previous data packet was not CE.

- MUST immediately send an ACK once 'n' CE marks have arrived since the previous ACK, where 'n' SHOULD be 2 and MUST be no greater than 6.

These rules for when to send an ACK are designed to be complemented by those in Section 3.2.3.3, which concern whether the AccECN TCP Option ought to be included on ACKs.

For the avoidance of doubt, the change-triggered ACK mechanism is deliberately worded to solely apply to data packets, and to ignore the arrival of a control packet with no payload, because it is important that TCP does not acknowledge pure ACKs. The change-triggered ACK approach can lead to some additional ACKs but it feeds back the timing and the order in which ECN marks are received with minimal additional complexity. If only CE marks are infrequent, or there are multiple marks in a row, the additional load will be low. Other marking patterns could increase the load significantly.

Even though the first bullet is stated as a "SHOULD", it is important for a transition to immediately trigger an ACK if at all possible, so that the Data Sender can rely on change-triggered ACKs to detect queue growth as soon as possible, e.g. at the start of a flow. This requirement can only be relaxed if certain offload hardware needed for high performance cannot support change-triggered ACKs (although high performance protocols such as DCTCP already successfully use change-triggered ACKs). One possible compromise would be for the receiver to heuristically detect whether the sender is in slow-start, then to implement change-triggered ACKs while the sender is in slow-start, and offload otherwise.

3.2.2.5.2. Data Sender Safety Procedures

If the Data Sender has not received AccECN TCP Options to give it more dependable information, and it detects that the ACE field could have cycled, it SHOULD deem whether it cycled by taking the safest likely case under the prevailing conditions. It can detect if the counter could have cycled by using the jump in the acknowledgement number since the last ACK to calculate or estimate how many segments could have been acknowledged. An example algorithm to implement this policy is given in Appendix A.2. An implementer MAY develop an alternative algorithm as long as it satisfies these requirements.

If missing acknowledgement numbers arrive later (reordering) and prove that the counter did not cycle, the Data Sender MAY attempt to neutralize the effect of any action it took based on a conservative assumption that it later found to be incorrect.

The Data Sender can estimate how many packets (of any marking) an ACK acknowledges. If the ACE counter on an ACK seems to imply that the minimum number of newly CE-marked packets is greater that the number of newly acknowledged packets, the Data Sender SHOULD believe the ACE counter, unless it can be sure that it is counting all control packets correctly.

3.2.3. The AccECN Option

The AccECN Option is defined as shown in Figure 4. The initial 'E' of each field name stands for 'Echo'.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Kind = TBD0  |  Length = 11  |          EE0B field           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | EE0B (cont’d) |           ECEB field                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                  EE1B field                   |       Order 0
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |  Kind = TBD1  |  Length = 11  |          EE1B field           |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   | EE1B (cont’d) |           ECEB field                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                  EE0B field                   |       Order 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: The AccECN TCP Option

Figure 4 shows two option field orders; order 0 and order 1. They both consists of three 24-bit fields. Order 0 provides the 24 least significant bits of the r.e0b, r.ceb and r.e1b counters, respectively. Order 1 provides the same fields, but in the opposite order. On each packet, the Data Receiver can use whichever order is more efficient.

When a Data Receiver sends an AccECN Option, it MUST set the Kind field to TBD0 if using Order 0, or to TBD1 if using Order 1. These two new TCP Option Kinds are registered in Section 7 and called respectively AccECN0 and AccECN1.

Note that there is no field to feed back Not-ECT bytes. Nonetheless an algorithm for the Data Sender to calculate the number of payload bytes received as Not-ECT is given in Appendix A.5.

Whenever a Data Receiver sends an AccECN Option, the rules in Section 3.2.3.3 expect it to usually send a full-length option. To cope with option space limitations, it can omit unchanged fields from the tail of the option, as long as it preserves the order of the remaining fields and includes any field that has changed. The length field MUST indicate which fields are present as follows:

Length	Type 0	Type 1
11	EE0B, ECEB, EE1B	EE1B, ECEB, EE0B
8	EE0B, ECEB	EE1B, ECEB
5	EE0B	EE1B
2	(empty)	(empty)

The empty option of Length=2 is provided to allow for a case where an AccECN Option has to be sent (e.g. on the SYN/ACK to test the path), but there is very limited space for the option.

All implementations of a Data Sender that read any AccECN Option MUST be able to read in AccECN Options of any of the above lengths. For forward compatibility, if the AccECN Option is of any other length, implementations MUST use those whole 3-octet fields that fit within the length and ignore the remainder of the option.

The AccECN Option has to be optional to implement, because both sender and receiver have to be able to cope without the option anyway - in cases where it does not traverse a network path. It is RECOMMENDED to implement both sending and receiving of the AccECN Option. If sending of the AccECN Option is implemented, the fall-backs described in this document will need to be implemented as well (unless solely for a controlled environment where path traversal is not considered a problem). Even if a developer does not implement sending of the AccECN Option, it is RECOMMENDED that they still implement logic to receive and understand any AccECN Options sent by remote peers.

If a Data Receiver intends to send the AccECN Option at any time during the rest of the connection it is strongly recommended to also test path traversal of the AccECN Option as specified in Section 3.2.3.2.

3.2.3.1. Encoding and Decoding Feedback in the AccECN Option Fields

Whenever the Data Receiver includes any of the counter fields (ECEB, EE0B, EE1B) in an AccECN Option, it MUST encode the 24 least significant bits of the current value of the associated counter into the field (respectively r.ceb, r.e0b, r.e1b).

Whenever the Data Sender receives ACK carrying an AccECN Option, it first checks whether the ACK has already been superseded by another ACK in which case it ignores the ECN feedback. If the ACK has not been superseded, the Data Sender MUST decode the fields in the AccECN Option as follows. For each field, it takes the least significant 24 bits of its associated local counter (s.ceb, s.e0b or s.e1b) and subtracts them from the counter in the associated field of the incoming AccECN Option (respectively ECEB, EE0B, EE1B), to work out the minimum positive increment it could apply to s.ceb, s.e0b or s.e1b (assuming the field in the option only wrapped at most once).

Appendix A.1 gives an example algorithm for the Data Receiver to encode its byte counters into the AccECN Option, and for the Data Sender to decode the AccECN Option fields into its byte counters.

Note that, as specified in Section 3.2, any data on the SYN (SYN=1, ACK=0) is not included in any of the locally held octet counters nor in the AccECN Option on the wire.

3.2.3.2. Path Traversal of the AccECN Option

3.2.3.2.1. Testing the AccECN Option during the Handshake

The TCP client MUST NOT include the AccECN TCP Option on the SYN. (A fall-back strategy for the loss of the SYN (possibly due to middlebox interference) is specified in Section 3.1.4.)

A TCP server that confirms its support for AccECN (in response to an AccECN SYN from the client as described in Section 3.1) SHOULD include an AccECN TCP Option on the SYN/ACK.

A TCP client that has successfully negotiated AccECN SHOULD include an AccECN Option in the first ACK at the end of the 3WHS. However, this first ACK is not delivered reliably, so the TCP client SHOULD also include an AccECN Option on the first data segment it sends (if it ever sends one).

A host MAY NOT include an AccECN Option in any of these three cases if it has cached knowledge that the packet would be likely to be blocked on the path to the other host if it included an AccECN Option.

3.2.3.2.2. Testing for Loss of Packets Carrying the AccECN Option

If after the normal TCP timeout the TCP server has not received an ACK to acknowledge its SYN/ACK, the SYN/ACK might just have been lost, e.g. due to congestion, or a middlebox might be blocking the AccECN Option. To expedite connection setup, the TCP server SHOULD retransmit the SYN/ACK repeating the same AE, CWR and ECE TCP flags as on the original SYN/ACK but with no AccECN Option. If this retransmission times out, to expedite connection setup, the TCP server SHOULD disable AccECN and ECN for this connection by retransmitting the SYN/ACK with AE=CWR=ECE=0 and no AccECN Option.

Implementers MAY use other fall-back strategies if they are found to be more effective (e.g. retrying the AccECN Option for a second time before fall-back - most appropriate during high levels of congestion). However, other fall-back strategies will need to follow all the rules in Section 3.1.5, which concern behaviour when SYNs or SYN/ACKs negotiating different types of feedback have been sent within the same connection.

If the TCP client detects that the first data segment it sent with the AccECN Option was lost, it SHOULD fall back to no AccECN Option on the retransmission. Again, implementers MAY use other fall-back strategies such as attempting to retransmit a second segment with the AccECN Option before fall-back, and/or caching whether the AccECN Option is blocked for subsequent connections. [I-D.ietf-tcpm-2140bis] further discusses caching of TCP parameters and status information.

If a host falls back to not sending the AccECN Option, it will continue to process any incoming AccECN Options as normal.

Either host MAY include the AccECN Option in a subsequent segment to retest whether the AccECN Option can traverse the path.

If the TCP server receives a second SYN with a request for AccECN support, it should resend the SYN/ACK, again confirming its support for AccECN, but this time without the AccECN Option. This approach rules out any interference by middleboxes that may drop packets with unknown options, even though it is more likely that the SYN/ACK would have been lost due to congestion. The TCP server MAY try to send another packet with the AccECN Option at a later point during the connection but should monitor if that packet got lost as well, in which case it SHOULD disable the sending of the AccECN Option for this half-connection.

Similarly, an AccECN end-point MAY separately memorize which data packets carried an AccECN Option and disable the sending of AccECN Options if the loss probability of those packets is significantly higher than that of all other data packets in the same connection.

3.2.3.2.3. Testing for Absence of the AccECN Option

If the TCP client has successfully negotiated AccECN but does not receive an AccECN Option on the SYN/ACK (e.g. because is has been stripped by a middlebox or not sent by the server), the client switches into a mode that assumes that the AccECN Option is not available for this half connection.

Similarly, if the TCP server has successfully negotiated AccECN but does not receive an AccECN Option on the first segment that acknowledges sequence space at least covering the ISN, it switches into a mode that assumes that the AccECN Option is not available for this half connection.

While a host is in this mode that assumes incoming AccECN Options are not available, it MUST adopt the conservative interpretation of the ACE field discussed in Section 3.2.2.5. However, it cannot make any assumption about support of outgoing AccECN Options on the other half connection, so it SHOULD continue to send the AccECN Option itself (unless it has established that sending the AccECN Option is causing packets to be blocked as in Section 3.2.3.2.2).

If a host is in the mode that assumes incoming AccECN Options are not available, but it receives an AccECN Option at any later point during the connection, this clearly indicates that the AccECN Option is not blocked on the respective path, and the AccECN endpoint MAY switch out of the mode that assumes the AccECN Option is not available for this half connection.

3.2.3.2.4. Test for Zeroing of the AccECN Option

For a related test for invalid initialization of the ACE field, see Section 3.2.2.3

Section 3.2 required the Data Receiver to initialize the r.e0b counter to a non-zero value. Therefore, in either direction the initial value of the EE0B field in the AccECN Option (if one exists) ought to be non-zero. If AccECN has been negotiated:

- the TCP server MAY check the initial value of the EE0B field in the first segment that acknowledges sequence space that at least covers the ISN plus 1. If the initial value of the EE0B field is zero, the server will switch into a mode that ignores the AccECN Option for this half connection.

- the TCP client MAY check the initial value of the EE0B field on the SYN/ACK. If the initial value of the EE0B field is zero, the client will switch into a mode that ignores the AccECN Option for this half connection.

While a host is in the mode that ignores the AccECN Option it MUST adopt the conservative interpretation of the ACE field discussed in Section 3.2.2.5.

Note that the Data Sender MUST NOT test whether the arriving byte counters in the initial AccECN Option have been initialized to specific valid values - the above checks solely test whether these fields have been incorrectly zeroed. This allows hosts to use different initial values as an additional signalling channel in future. Also note that the initial value of either field might be greater than its expected initial value, because the counters might already have been incremented. Nonetheless, the initial values of the counters have been chosen so that they cannot wrap to zero on these initial segments.

3.2.3.2.5. Consistency between AccECN Feedback Fields

When the AccECN Option is available it supplements but does not replace the ACE field. An endpoint using AccECN feedback MUST always consider the information provided in the ACE field whether or not the AccECN Option is also available.

If the AccECN option is present, the s.cep counter might increase while the s.ceb counter does not (e.g. due to a CE-marked control packet). The sender’s response to such a situation is out of scope, and needs to be dealt with in a specification that uses ECN-capable control packets. Theoretically, this situation could also occur if a middlebox mangled the AccECN Option but not the ACE field. However, the Data Sender has to assume that the integrity of the AccECN Option is sound, based on the above test of the well-known initial values and optionally other integrity tests (Section 5.3).

If either end-point detects that the s.ceb counter has increased but the s.cep has not (and by testing ACK coverage it is certain how much the ACE field has wrapped), this invalid protocol transition has to be due to some form of feedback mangling. So, the Data Sender MUST disable sending ECN-capable packets for the remainder of the half-connection by setting the IP/ECN field in all subsequent packets to Not-ECT.

3.2.3.3. Usage of the AccECN TCP Option

If the Data Receiver intends to use the AccECN TCP Option to provide feedback, the following rules determine when a Data Receiver in AccECN mode sends an ACK with the AccECN TCP Option, and which fields to include:

Change-Triggered ACKs: If an arriving packet increments a different byte counter to that incremented by the previous packet, the Data Receiver SHOULD immediately send an ACK with an AccECN Option, without waiting for the next delayed ACK (this is in addition to the safety recommendation in Section 3.2.2.5 against ambiguity of the ACE field).

Even though this bullet is stated as a "SHOULD", it is important for a transition to immediately trigger an ACK if at all possible, as already argued when specifying change-triggered ACKs for the ACE.

Continual Repetition: Otherwise, if arriving packets continue to increment the same byte counter, the Data Receiver can include an AccECN Option on most or all (delayed) ACKs, but it does not have to.

* It SHOULD include a counter that has continued to increment on the next scheduled ACK following a change-triggered ACK;

* while the same counter continues to increment, it SHOULD include the counter every n ACKs as consistently as possible, where n can be chosen by the implementer;

* It SHOULD always include an AccECN Option if the r.ceb counter is incrementing and it MAY include an AccECN Option if r.ec0b or r.ec1b is incrementing

* It SHOULD, include each counter at least once for every 2^22 bytes incremented to prevent overflow during continual repetition.

If the smallest allowed AccECN Option would leave insufficient space for two SACK blocks on a particular ACK, the Data Receiver MUST give precedence to the SACK option (total 18 octets), because loss feedback is more critical.

Necessary Option Length: It MAY exclude counter(s) that have not changed for the whole connection (but beacons still include all fields - see below). It SHOULD include counter(s) that have incremented at some time during the connection. It MUST include the counter(s) that have incremented since the previous AccECN Option and it MUST only truncate fields from the right-hand tail of the option to preserve the order of the remaining fields (see Section 3.2.3);

Beaconing Full-Length Options: Nonetheless, it MUST include a full-length AccECN TCP Option on at least three ACKs per RTT, or on all ACKs if there are less than three per RTT (see Appendix A.4 for an example algorithm that satisfies this requirement).

The above rules complement those in Section 3.2.2.5, which determine when to generate an ACK irrespective of whether an AccECN TCP Option is to be included.

The following example series of arriving IP/ECN fields illustrates when a Data Receiver will emit an ACK with an AccECN Option if it is using a delayed ACK factor of 2 segments and change-triggered ACKs: 01 -> ACK, 01, 01 -> ACK, 10 -> ACK, 10, 01 -> ACK, 01, 11 -> ACK, 01 -> ACK.

Even though first bullet is stated as a "SHOULD", it is important for a transition to immediately trigger an ACK if at all possible, so that the Data Sender can rely on change-triggered ACKs to detect queue growth as soon as possible, e.g. at the start of a flow. This requirement can only be relaxed if certain offload hardware needed for high performance cannot support change-triggered ACKs (although high performance protocols such as DCTCP already successfully use change-triggered ACKs). One possible experimental compromise would be for the receiver to heuristically detect whether the sender is in slow-start, then to implement change-triggered ACKs while the sender is in slow-start, and offload otherwise.

For the avoidance of doubt, this change-triggered ACK mechanism is deliberately worded to ignore the arrival of a control packet with no payload, which therefore does not alter any byte counters, because it is important that TCP does not acknowledge pure ACKs. The change-triggered ACK approach can lead to some additional ACKs but it feeds back the timing and the order in which ECN marks are received with minimal additional complexity. If only CE marks are infrequent, or there are multiple marks in a row, the additional load will be low. Other marking patterns could increase the load significantly, Investigating the additional load is a goal of the proposed experiment.

Implementation note: sending an AccECN Option each time a different counter changes and including a full-length AccECN Option on every delayed ACK will satisfy the requirements described above and might be the easiest implementation, as long as sufficient space is available in each ACK (in total and in the option space).

Appendix A.3 gives an example algorithm to estimate the number of marked bytes from the ACE field alone, if the AccECN Option is not available.

If a host has determined that segments with the AccECN Option always seem to be discarded somewhere along the path, it is no longer obliged to follow the above rules.

3.3. AccECN Compliance Requirements for TCP Proxies, Offload Engines and other Middleboxes

3.3.1. Requirements for TCP Proxies

A large class of middleboxes split TCP connections. Such a middlebox would be compliant with the AccECN protocol if the TCP implementation on each side complied with the present AccECN specification and each side negotiated AccECN independently of the other side.

3.3.2. Requirements for TCP Normalizers

Another large class of middleboxes intervenes to some degree at the transport layer, but attempts to be transparent (invisible) to the end-to-end connection. A subset of this class of middleboxes attempts to ‘normalize’ the TCP wire protocol by checking that all values in header fields comply with a rather narrow and often outdated interpretation of the TCP specifications. To comply with the present AccECN specification, such a middlebox MUST NOT change the ACE field or the AccECN Option.

A middlebox claiming to be transparent at the transport layer MUST forward the AccECN TCP Option unaltered, whether or not the length value matches one of those specified in Section 3.2.3, and whether or not the initial values of the byte-counter fields are correct. This is because blocking apparently invalid values does not improve security (because AccECN hosts are required to ignore invalid values anyway), while it prevents the standardized set of values being extended in future (because outdated normalizers would block updated hosts from using the extended AccECN standard).

3.3.3. Requirements for TCP ACK Filtering

A node that implements ACK filtering (aka. thinning or coalescing) and itself also implements ECN marking will not need to filter ACKs from connections that use AccECN feedback. Therefore, such a node SHOULD detect connections that have negotiated the use of AccECN feedback during the handshake (see Table 2) and it SHOULD preserve the timing of each ACK (if it coalesced ACKs it would not be AccECN-compliant, but the requirement is stated as a "SHOULD" in order to allow leeway for pre-existing ACK filtering functions to be brought into line).

A node that implements ACK filtering and does not itself implement ECN marking does not need to treat AccECN connections any differently from other TCP connections. Nonetheless, it is RECOMMENDED that such nodes implement ECN marking and comply with the requirements of the previoius paragraph. This should be a better way than ACK filtering to improve the performance of AccECN TCP connections.

The rationale for these requirements is that AccECN feedback provides sufficient information to a data receiver for it to be able to monitor ECN marking of the ACKs it has sent, so that it can thin the ACK stream itself. This will eventually mean that ACK filtering in the network gives no performance advantage. Then TCP will be able to maintain its own control over ACK coalescing. This will also allow the TCP Data Sender to use the timing of ACK arrivals to more reliably infer further information about the path congestion level.

Note that the specification of AccECN in TCP does not presume to rely on the above ACK filtering behaviour in the network, because it has to be robust against pre-existing network nodes that still filter AccECN ACKs, and robust against ACK loss during overload.

Section 5.2.1 of [RFC3449] gives best current practice on ACK filtering (aka. thinning or coalescing). It gives no advice on ACKs carrying ECN feedback, because at the time is said that "ECN remain areas of ongoing research". This section updates that advice for a TCP connection that supports AccECN feedback.

3.3.4. Requirements for TCP Segmentation Offload

Hardware to offload certain TCP processing represents another large class of middleboxes (even though it is often a function of a host's network interface and rarely in its own 'box').

The ACE field changes with every received CE marking, so today's receive offloading could lead to many interrupts in high congestion situations. Although that would be useful (because congestion information is received sooner), it could also significantly increase processor load, particularly in scenarios such as DCTCP or L4S where the marking rate is generally higher.

Current offload hardware ejects a segment from the coalescing process whenever the TCP ECN flags change. Thus Classic ECN causes offload to be inefficient. In data centres it has been fortunate for this offload hardware that DCTCP-style feedback changes less often when there are long sequences of CE marks, which is more common with a step marking threshold (but less likely the more short flows are in the mix). The ACE counter approach has been designed so that coalescing can continue over arbitrary patterns of marking and only needs to stop when the counter wraps. Nonetheless, until the particular offload hardware in use implements this more efficient approach, it is likely to be more efficient for AccECN connections to implement this counter-style logic using software segmentation offload.

ECN encodes a varying signal in the ACK stream, so it is inevitable that offload hardware will ultimately need to handle any form of ECN feedback exceptionally. The ACE field has been designed as a counter so that it is straightforward for offload hardware to pass on the highest counter, and to push a segment from its cache before the counter wraps. The purpose of working towards standardized TCP ECN feedback is to reduce the risk for hardware developers, who would otherwise have to guess which scheme is likely to become dominant.

The above process has been designed to enable a continuing incremental deployment path - to more highly dynamic congestion control. Once DCTCP offload hardware supports AccECN, it will be able to coalesce efficiently for any sequence of marks, instead of relying for efficiency on the long marking sequences from step marking. In the next stage, DCTCP marking can evolve from a step to a ramp function. That in turn will allow host congestion control algorithms to respond faster to dynamics, while being backwards compatible with existing host algorithms.

4. Updates to RFC 3168

Normative statements in the following sections of RFC3168 are updated by the present AccECN specification:

- The whole of "6.1.1 TCP Initialization" of [RFC3168] is updated by Section 3.1 of the present specification.

- In "6.1.2. The TCP Sender" of [RFC3168], all mentions of a congestion response to an ECN-Echo (ECE) ACK packet are updated by Section 3.2 of the present specification to mean an increment to the sender's count of CE-marked packets, s.cep. And the requirements to set the CWR flag no longer apply, as specified in Section 3.1.5 of the present specification. Otherwise, the remaining requirements in "6.1.2. The TCP Sender" still stand.

  It will be noted that RFC 8311 already updates, or potentially updates, a number of the requirements in "6.1.2. The TCP Sender". Section 6.1.2 of RFC 3168 extended standard TCP congestion control [RFC5681] to cover ECN marking as well as packet drop. Whereas, RFC 8311 enables experimentation with alternative responses to ECN marking, if specified for instance by an experimental RFC on the IETF document stream. RFC 8311 also strengthened the statement that "ECT(0) SHOULD be used" to a "MUST" (see [RFC8311] for the details).

- The whole of "6.1.3. The TCP Receiver" of [RFC3168] is updated by Section 3.2 of the present specification, with the exception of the last paragraph (about congestion response to drop and ECN in the same round trip), which still stands. Incidentally, this last paragraph is in the wrong section, because it relates to TCP sender behaviour.

- The following text within "6.1.5. Retransmitted TCP packets":

  > "the TCP data receiver SHOULD ignore the ECN field on arriving data packets that are outside of the receiver's current window."

  is updated by more stringent acceptability tests for any packet (not just data packets) in the present specification. Specifically, in the normative specification of AccECN (Section 3) only 'Acceptable' packets contribute to the ECN counters at the AccECN receiver and Section 1.3 defines an Acceptable packet as one that passes the acceptability tests in both [RFC0793] and [RFC5961].

- Sections 5.2, 6.1.1, 6.1.4, 6.1.5 and 6.1.6 of [RFC3168] prohibit use of ECN on TCP control packets and retransmissions. The present specification does not update that aspect of RFC 3168, but it does say what feedback an AccECN Data Receiver should provide if it receives an ECN-capable control packet or retransmission. This ensures AccECN is forward compatible with any future scheme that allows ECN on these packets, as provided for in section 4.3 of [RFC8311] and as proposed in [I-D.ietf-tcpm-generalized-ecn].

5. Interaction with TCP Variants

This section is informative, not normative.

5.1. Compatibility with SYN Cookies

A TCP server can use SYN Cookies (see Appendix A of [RFC4987]) to protect itself from SYN flooding attacks. It places minimal commonly used connection state in the SYN/ACK, and deliberately does not hold any state while waiting for the subsequent ACK (e.g. it closes the thread). Therefore it cannot record the fact that it entered AccECN mode for both half-connections. Indeed, it cannot even remember whether it negotiated the use of classic ECN [RFC3168].

Nonetheless, such a server can determine that it negotiated AccECN as follows. If a TCP server using SYN Cookies supports AccECN and if it receives a pure ACK that acknowledges an ISN that is a valid SYN cookie, and if the ACK contains an ACE field with the value 0b010 to 0b111 (decimal 2 to 7), it can assume that:

- the TCP client must have requested AccECN support on the SYN

- it (the server) must have confirmed that it supported AccECN

Therefore the server can switch itself into AccECN mode, and continue as if it had never forgotten that it switched itself into AccECN mode earlier.

If the pure ACK that acknowledges a SYN cookie contains an ACE field with the value 0b000 or 0b001, these values indicate that the client did not request support for AccECN and therefore the server does not enter AccECN mode for this connection. Further, 0b001 on the ACK implies that the server sent an ECN-capable SYN/ACK, which was marked CE in the network, and the non-AccECN client fed this back by setting ECE on the ACK of the SYN/ACK.

5.2. Compatibility with TCP Experiments and Common TCP Options

AccECN is compatible (at least on paper) with the most commonly used TCP options: MSS, time-stamp, window scaling, SACK and TCP-AO. It is also compatible with the recent promising experimental TCP options TCP Fast Open (TFO [RFC7413]) and Multipath TCP (MPTCP [RFC6824]). AccECN is friendly to all these protocols, because space for TCP options is particularly scarce on the SYN, where AccECN consumes zero additional header space.

When option space is under pressure from other options, Section 3.2.3.3 provides guidance on how important it is to send an AccECN Option and whether it needs to be a full-length option.

Implementers of TFO need to take careful note of the recommendation in Section 3.2.2.1. That section recommends that, if the client has successfully negotiated AccECN, when acknowledging the SYN/ACK, even if it has data to send, it sends a pure ACK immediately before the data. Then it can reflect the IP-ECN field of the SYN/ACK on this pure ACK, which allows the server to detect ECN mangling.

5.3. Compatibility with Feedback Integrity Mechanisms

Three alternative mechanisms are available to assure the integrity of ECN and/or loss signals. AccECN is compatible with any of these approaches:

- The Data Sender can test the integrity of the receiver's ECN (or loss) feedback by occasionally setting the IP-ECN field to a value normally only set by the network (and/or deliberately leaving a sequence number gap). Then it can test whether the Data Receiver's feedback faithfully reports what it expects (similar to para 2 of Section 20.2 of [RFC3168]). Unlike the ECN Nonce [RFC3540], this approach does not waste the ECT(1) codepoint in the IP header, it does not require standardization and it does not rely on misbehaving receivers volunteering to reveal feedback information that allows them to be detected. However, setting the CE mark by the sender might conceal actual congestion feedback from the network and should therefore only be done sparingly.

- Networks generate congestion signals when they are becoming congested, so networks are more likely than Data Senders to be concerned about the integrity of the receiver's feedback of these signals. A network can enforce a congestion response to its ECN markings (or packet losses) using congestion exposure (ConEx) audit [RFC7713]. Whether the receiver or a downstream network is suppressing congestion feedback or the sender is unresponsive to the feedback, or both, ConEx audit can neutralize any advantage that any of these three parties would otherwise gain.

  ConEx is a change to the Data Sender that is most useful when combined with AccECN. Without AccECN, the ConEx behaviour of a Data Sender would have to be more conservative than would be necessary if it had the accurate feedback of AccECN.

- The TCP authentication option (TCP-AO [RFC5925]) can be used to detect any tampering with AccECN feedback between the Data Receiver and the Data Sender (whether malicious or accidental). The AccECN fields are immutable end-to-end, so they are amenable to TCP-AO protection, which covers TCP options by default. However, TCP-AO is often too brittle to use on many end-to-end paths, where middleboxes can make verification fail in their attempts to improve performance or security, e.g. by resegmentation or shifting the sequence space.

Originally the ECN Nonce [RFC3540] was proposed to ensure integrity of congestion feedback. With minor changes AccECN could be optimized for the possibility that the ECT(1) codepoint might be used as an ECN Nonce. However, given RFC 3540 has been reclassified as historic, the AccECN design has been generalized so that it ought to be able to support other possible uses of the ECT(1) codepoint, such as a lower severity or a more instant congestion signal than CE.

6. Protocol Properties

This section is informative not normative. It describes how well the protocol satisfies the agreed requirements for a more accurate ECN feedback protocol [RFC7560].

Accuracy: From each ACK, the Data Sender can infer the number of new CE marked segments since the previous ACK. This provides better accuracy on CE feedback than classic ECN. In addition if the AccECN Option is present (not blocked by the network path) the number of bytes marked with CE, ECT(1) and ECT(0) are provided.

Overhead: The AccECN scheme is divided into two parts. The essential part reuses the 3 flags already assigned to ECN in the IP header. The supplementary part adds an additional TCP option consuming up to 11 bytes. However, no TCP option is consumed in the SYN.

Ordering: The order in which marks arrive at the Data Receiver is preserved in AccECN feedback, because the Data Receiver is expected to send an ACK immediately whenever a different mark arrives.

Timeliness: While the same ECN markings are arriving continually at the Data Receiver, it can defer ACKs as TCP does normally, but it will immediately send an ACK as soon as a different ECN marking arrives.

Timeliness vs Overhead: Change-Triggered ACKs are intended to enable latency-sensitive uses of ECN feedback by capturing the timing of transitions but not wasting resources while the state of the signalling system is stable. Within the constraints of the change-triggered ACK rules, the receiver can control how frequently it sends the AccECN TCP Option and therefore to some extent it can control the overhead induced by AccECN.

Resilience: All information is provided based on counters. Therefore if ACKs are lost, the counters on the first ACK following the losses allows the Data Sender to immediately recover the number of the ECN markings that it missed. And if data or ACKs are reordered, stale congestion information can be identified and ignored.

Resilience against Bias: Because feedback is based on repetition of counters, random losses do not remove any information, they only delay it. Therefore, even though some ACKs are change-triggered, random losses will not alter the proportions of the different ECN markings in the feedback.

Resilience vs Overhead: If space is limited in some segments (e.g. because more options are needed on some segments, such as the SACK option after loss), the Data Receiver can send AccECN Options less frequently or truncate fields that have not changed, usually down to as little as 5 bytes. However, it has to send a full-sized AccECN Option at least three times per RTT, which the Data Sender can rely on as a regular beacon or checkpoint.

Resilience vs Timeliness and Ordering: Ordering information and the timing of transitions cannot be communicated in three cases: i) during ACK loss; ii) if something on the path strips the AccECN Option; or iii) if the Data Receiver is unable to support Change-Triggered ACKs. Following ACK reordering, the Data Sender can reconstruct the order in which feedback was sent, but not until all the missing feedback has arrived.

Complexity: An AccECN implementation solely involves simple counter increments, some modulo arithmetic to communicate the least significant bits and allow for wrap, and some heuristics for safety against fields cycling due to prolonged periods of ACK loss. Each host needs to maintain eight additional counters. The hosts have to apply some additional tests to detect tampering by middleboxes, but in general the protocol is simple to understand, simple to implement and requires few cycles per packet to execute.

Integrity: AccECN is compatible with at least three approaches that can assure the integrity of ECN feedback. If the AccECN Option is stripped the resolution of the feedback is degraded, but the integrity of this degraded feedback can still be assured.

Backward Compatibility: If only one endpoint supports the AccECN scheme, it will fall-back to the most advanced ECN feedback scheme supported by the other end.

Backward Compatibility: If the AccECN Option is stripped by a middlebox, AccECN still provides basic congestion feedback in the ACE field. Further, AccECN can be used to detect mangling of the IP ECN field; mangling of the TCP ECN flags; blocking of ECT-marked segments; and blocking of segments carrying the AccECN Option. It can detect these conditions during TCP’s 3WHS so that it can fall back to operation without ECN and/or operation without the AccECN Option.

Forward Compatibility: The behaviour of endpoints and middleboxes is carefully defined for all reserved or currently unused codepoints in the scheme. Then, the designers of security devices can understand which currently unused values might appear in future. So, even if they choose to treat such values as anomalous while they are not widely used, any blocking will at least be under policy control not hard-coded. Then, if previously unused values start to appear on the Internet (or in standards), such policies could be quickly reversed.

7. IANA Considerations

This document reassigns bit 7 of the TCP header flags to the AccECN experiment. This bit was previously called the Nonce Sum (NS) flag [RFC3540], but RFC 3540 has been reclassified as historic [RFC8311]. The flag will now be defined as:

Bit	Name	Reference
7	AE (Accurate ECN)	RFC XXXX

[TO BE REMOVED: IANA is requested to update the existing entry in the Transmission Control Protocol (TCP) Header Flags registration (https://www.iana.org/assignments/tcp-header-flags/tcp-header-flags.xhtml#tcp-header-flags-1) for Bit 7 to "AE (Accurate ECN), previously used as NS (Nonce Sum) by [RFC3540], which is now Historic [RFC8311]" and change the reference to this RFC-to-be instead of RFC8311.]

This document also defines two new TCP options for AccECN, assigned values of TBD0 and TBD1 (decimal) from the TCP option space. These values are defined as:

Kind	Length	Meaning	Reference
TBD0	N	Accurate ECN Order 0 (AccECN0)	RFC XXXX
TBD1	N	Accurate ECN Order 1 (AccECN1)	RFC XXXX

[TO BE REMOVED: This registration should take place at the following location: http://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-parameters-1]

Early implementations using experimental option 254 per [RFC6994] with the single magic number 0xACCE (16 bits), as allocated in the IANA "TCP Experimental Option Experiment Identifiers (TCP ExIDs)" registry, SHOULD migrate to use these new option kinds (TBD0 & TBD1).

[TO BE REMOVED: The description of the 0xACCE value in the TCP ExIDs registry should be changed to "AccECN (current and new implementations SHOULD use option kinds TBD0 and TBD1)" at the following location: https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml#tcp-exids]

8. Security Considerations

If ever the supplementary part of AccECN based on the new AccECN TCP Option is unusable (due for example to middlebox interference) the essential part of AccECN's congestion feedback offers only limited resilience to long runs of ACK loss (see Section 3.2.2.5). These problems are unlikely to be due to malicious intervention (because if an attacker could strip a TCP option or discard a long run of ACKs it could wreak other arbitrary havoc). However, it would be of concern if AccECN's resilience could be indirectly compromised during a flooding attack. AccECN is still considered safe though, because if the option is not presented, the AccECN Data Sender is then required to switch to more conservative assumptions about wrap of congestion indication counters (see Section 3.2.2.5 and Appendix A.2).

Section 5.1 describes how a TCP server can negotiate AccECN and use the SYN cookie method for mitigating SYN flooding attacks.

There is concern that ECN markings could be altered or suppressed, particularly because a misbehaving Data Receiver could increase its own throughput at the expense of others. AccECN is compatible with the three schemes known to assure the integrity of ECN feedback (see Section 5.3 for details). If the AccECN Option is stripped by an incorrectly implemented middlebox, the resolution of the feedback will be degraded, but the integrity of this degraded information can still be assured.

There is a potential concern that a receiver could deliberately omit the AccECN Option pretending that it had been stripped by a middlebox. No known way can yet be contrived to take advantage of this downgrade attack, but it is mentioned here in case someone else can contrive one.

The AccECN protocol is not believed to introduce any new privacy concerns, because it merely counts and feeds back signals at the transport layer that had already been visible at the IP layer.

9. Acknowledgements

We want to thank Koen De Schepper, Praveen Balasubramanian, Michael Welzl, Gorry Fairhurst, David Black, Spencer Dawkins, Michael Scharf, Michael Tuexen, Yuchung Cheng, Kenjiro Cho, Olivier Tilmans, Ilpo Jaervinen and Neal Cardwell for their input and discussion. The idea of using the three ECN-related TCP flags as one field for more accurate TCP-ECN feedback was first introduced in the re-ECN protocol that was the ancestor of ConEx.

Bob Briscoe was part-funded by the Comcast Innovation Fund, the European Community under its Seventh Framework Programme through the Reducing Internet Transport Latency (RITE) project (ICT-317700) and through the Trilogy 2 project (ICT-317756), and the Research Council of Norway through the TimeIn project. The views expressed here are solely those of the authors.

Mirja Kuehlewind was partly supported by the European Commission under Horizon 2020 grant agreement no. 688421 Measurement and Architecture for a Middleboxed Internet (MAMI), and by the Swiss State Secretariat for Education, Research, and Innovation under contract no. 15.0268. This support does not imply endorsement.

10. Comments Solicited

Comments and questions are encouraged and very welcome. They can be addressed to the IETF TCP maintenance and minor modifications working group mailing list <tcpm@ietf.org>, and/or to the authors.

11. References

11.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

11.2. Informative References

[I-D.ietf-tcpm-2140bis]
Touch, J., Welzl, M., and S. Islam, "TCP Control Block Interdependence", draft-ietf-tcpm-2140bis-05 (work in progress), April 2020.

[I-D.ietf-tcpm-generalized-ecn]
Bagnulo, M. and B. Briscoe, "ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets", draft-ietf-tcpm-generalized-ecn-06 (work in progress), October 2020.

[I-D.ietf-tsvwg-l4s-arch]
Briscoe, B., Schepper, K., Bagnulo, M., and G. White, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", draft-ietf-tsvwg-l4s-arch-07 (work in progress), October 2020.

[Mandalari18]
Mandalari, A., Lutu, A., Briscoe, B., Bagnulo, M., and Oe. Alay, "Measuring ECN++: Good News for ++, Bad News for ECN over Mobile", IEEE Communications Magazine , March 2018.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <https://www.rfc-editor.org/info/rfc2018>.

[RFC3449] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <https://www.rfc-editor.org/info/rfc3449>.

[RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <https://www.rfc-editor.org/info/rfc3540>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <https://www.rfc-editor.org/info/rfc4987>.

[RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <https://www.rfc-editor.org/info/rfc5562>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <https://www.rfc-editor.org/info/rfc5925>.

[RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <https://www.rfc-editor.org/info/rfc5961>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <https://www.rfc-editor.org/info/rfc6994>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[RFC7560] Kuehlewind, M., Ed., Scheffenegger, R., and B. Briscoe, "Problem Statement and Requirements for Increased Accuracy in Explicit Congestion Notification (ECN) Feedback", RFC 7560, DOI 10.17487/RFC7560, August 2015, <https://www.rfc-editor.org/info/rfc7560>.

[RFC7713] Mathis, M. and B. Briscoe, "Congestion Exposure (ConEx) Concepts, Abstract Mechanism, and Requirements", RFC 7713, DOI 10.17487/RFC7713, December 2015, <https://www.rfc-editor.org/info/rfc7713>.

[RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <https://www.rfc-editor.org/info/rfc8257>.

[RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <https://www.rfc-editor.org/info/rfc8311>.

[RFC8511] Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst, "TCP Alternative Backoff with ECN (ABE)", RFC 8511, DOI 10.17487/RFC8511, December 2018, <https://www.rfc-editor.org/info/rfc8511>.

Appendix A. Example Algorithms

This appendix is informative, not normative. It gives example algorithms that would satisfy the normative requirements of the AccECN protocol. However, implementers are free to choose other ways to implement the requirements.

A.1. Example Algorithm to Encode/Decode the AccECN Option

The example algorithms below show how a Data Receiver in AccECN mode could encode its CE byte counter r.ceb into the ECEB field within the AccECN TCP Option, and how a Data Sender in AccECN mode could decode the ECEB field into its byte counter s.ceb. The other counters for bytes marked ECT(0) and ECT(1) in the AccECN Option would be similarly encoded and decoded.

It is assumed that each local byte counter is an unsigned integer greater than 24b (probably 32b), and that the following constant has been assigned:

```
DIVOPT = 2^24
```

Every time a CE marked data segment arrives, the Data Receiver increments its local value of r.ceb by the size of the TCP Data. Whenever it sends an ACK with the AccECN Option, the value it writes into the ECEB field is

```
ECEB = r.ceb % DIVOPT
```

where '%' is the remainder operator.

On the arrival of an AccECN Option, the Data Sender first makes sure the ACK has not been superseded in order to avoid winding the s.ceb counter backwards. It uses the TCP acknowledgement number and any SACK options to calculate newlyAckedB, the amount of new data that the ACK acknowledges in bytes (newlyAckedB can be zero but not negative). If newlyAckedB is zero, either the ACK has been superseded or CE-marked packet(s) without data could have arrived. To break the tie for the latter case, the Data Sender could use timestamps (if present) to work out newlyAckedT, the amount of new time that the ACK acknowledges. If the Data Sender determines that the ACK has been superseded it ignores the AccECN Option. Otherwise, the Data Sender calculates the minimum non-negative difference d.ceb between the ECEB field and its local s.ceb counter, using modulo arithmetic as follows:

```
      if ((newlyAckedB > 0) || (newlyAckedT > 0)) {
          d.ceb = (ECEB + DIVOPT - (s.ceb % DIVOPT)) % DIVOPT
          s.ceb += d.ceb
      }
```

For example, if s.ceb is 33,554,433 and ECEB is 1461 (both decimal), then

```
      s.ceb % DIVOPT = 1
            d.ceb = (1461 + 2^24 - 1) % 2^24
                  = 1460
            s.ceb = 33,554,433 + 1460
                  = 33,555,893
```

A.2. Example Algorithm for Safety Against Long Sequences of ACK Loss

The example algorithms below show how a Data Receiver in AccECN mode could encode its CE packet counter r.cep into the ACE field, and how the Data Sender in AccECN mode could decode the ACE field into its s.cep counter. The Data Sender's algorithm includes code to heuristically detect a long enough unbroken string of ACK losses that could have concealed a cycle of the congestion counter in the ACE field of the next ACK to arrive.

Two variants of the algorithm are given: i) a more conservative variant for a Data Sender to use if it detects that the AccECN Option is not available (see Section 3.2.2.5 and Section 3.2.3.2); and ii) a less conservative variant that is feasible when complementary information is available from the AccECN Option.

A.2.1. Safety Algorithm without the AccECN Option

It is assumed that each local packet counter is a sufficiently sized unsigned integer (probably 32b) and that the following constant has been assigned:

```
      DIVACE = 2^3
```

Every time an Acceptable CE marked packet arrives (Section 3.2.2.2), the Data Receiver increments its local value of r.cep by 1. It repeats the same value of ACE in every subsequent ACK until the next CE marking arrives, where

```
      ACE = r.cep % DIVACE.
```

If the Data Sender received an earlier value of the counter that had been delayed due to ACK reordering, it might incorrectly calculate that the ACE field had wrapped. Therefore, on the arrival of every

ACK, the Data Sender ensures the ACK has not been superseded using the TCP acknowledgement number, any SACK options and timestamps (if available) to calculate newlyAckedB, as in Appendix A.1. If the ACK has not been superseded, the Data Sender calculates the minimum difference d.cep between the ACE field and its local s.cep counter, using modulo arithmetic as follows:

```
   if ((newlyAckedB > 0) || (newlyAckedT > 0))
       d.cep = (ACE + DIVACE - (s.cep % DIVACE)) % DIVACE
```

Section 3.2.2.5 expects the Data Sender to assume that the ACE field cycled if it is the safest likely case under prevailing conditions. The 3-bit ACE field in an arriving ACK could have cycled and become ambiguous to the Data Sender if a row of ACKs goes missing that covers a stream of data long enough to contain 8 or more CE marks. We use the word 'missing' rather than 'lost', because some or all the missing ACKs might arrive eventually, but out of order. Even if some of the missing ACKs were piggy-backed on data (i.e. not pure ACKs) retransmissions will not repair the lost AccECN information, because AccECN requires retransmissions to carry the latest AccECN counters, not the original ones.

The phrase 'under prevailing conditions' allows for implementation-dependent interpretation. A Data Sender might take account of the prevailing size of data segments and the prevailing CE marking rate just before the sequence of missing ACKs. However, we shall start with the simplest algorithm, which assumes segments are all full-sized and ultra-conservatively it assumes that ECN marking was 100% on the forward path when ACKs on the reverse path started to all be dropped. Specifically, if newlyAckedB is the amount of data that an ACK acknowledges since the previous ACK, then the Data Sender could assume that this acknowledges newlyAckedPkt full-sized segments, where newlyAckedPkt = newlyAckedB/MSS. Then it could assume that the ACE field incremented by

```
    dSafer.cep = newlyAckedPkt - ((newlyAckedPkt - d.cep) % DIVACE),
```

For example, imagine an ACK acknowledges newlyAckedPkt=9 more full-size segments than any previous ACK, and that ACE increments by a minimum of 2 CE marks (d.cep=2). The above formula works out that it would still be safe to assume 2 CE marks (because 9 - ((9-2) % 8) = 2). However, if ACE increases by a minimum of 2 but acknowledges 10 full-sized segments, then it would be necessary to assume that there could have been 10 CE marks (because 10 - ((10-2) % 8) = 10).

ACKs that acknowledge a large stretch of packets might be common in data centres to achieve a high packet rate or might be due to ACK thinning by a middlebox. In these cases, cycling of the ACE field would often appear to have been possible, so the above algorithm would be over-conservative, leading to a false high marking rate and poor performance. Therefore it would be reasonable to only use dSafer.cep rather than d.cep if the moving average of newlyAckedPkt was well below 8.

Implementers could build in more heuristics to estimate prevailing average segment size and prevailing ECN marking. For instance, newlyAckedPkt in the above formula could be replaced with newlyAckedPktHeur = newlyAckedPkt*p*MSS/s, where s is the prevailing segment size and p is the prevailing ECN marking probability. However, ultimately, if TCP's ECN feedback becomes inaccurate it still has loss detection to fall back on. Therefore, it would seem safe to implement a simple algorithm, rather than a perfect one.

The simple algorithm for dSafer.cep above requires no monitoring of prevailing conditions and it would still be safe if, for example, segments were on average at least 5% of full-sized as long as ECN marking was 5% or less. Assuming it was used, the Data Sender would increment its packet counter as follows:

```
s.cep += dSafer.cep
```

If missing acknowledgement numbers arrive later (due to reordering), Section 3.2.2.5 says "the Data Sender MAY attempt to neutralize the effect of any action it took based on a conservative assumption that it later found to be incorrect". To do this, the Data Sender would have to store the values of all the relevant variables whenever it made assumptions, so that it could re-evaluate them later. Given this could become complex and it is not required, we do not attempt to provide an example of how to do this.

A.2.2. Safety Algorithm with the AccECN Option

When the AccECN Option is available on the ACKs before and after the possible sequence of ACK losses, if the Data Sender only needs CE-marked bytes, it will have sufficient information in the AccECN Option without needing to process the ACE field. If for some reason it needs CE-marked packets, if dSafer.cep is different from d.cep, it can determine whether d.cep is likely to be a safe enough estimate by checking whether the average marked segment size (s = d.ceb/d.cep) is less than the MSS (where d.ceb is the amount of newly CE-marked bytes - see Appendix A.1). Specifically, it could use the following algorithm:

```
   SAFETY_FACTOR = 2
   if (dSafer.cep > d.cep) {
       if (d.ceb <= MSS * d.cep) {  % Same as (s <= MSS), but no DBZ
          sSafer = d.ceb/dSafer.cep
          if (sSafer < MSS/SAFETY_FACTOR)
              dSafer.cep = d.cep    % d.cep is a safe enough estimate
       } % else
           % No need for else; dSafer.cep is already correct,
           % because d.cep must have been too small
   }
```

The chart below shows when the above algorithm will consider d.cep can replace dSafer.cep as a safe enough estimate of the number of CE-marked packets:

```
                  ^
            sSafer|
                  |
               MSS+
                  |
                  |         dSafer.cep
                  |                  is
MSS/SAFETY_FACTOR+---------------+    safest
                  |               |
                  | d.cep is safe|
                  |    enough     |
                  +-------------------->
                            MSS     s
```

The following examples give the reasoning behind the algorithm, assuming MSS=1460 [B]:

- if d.cep=0, dSafer.cep=8 and d.ceb=1460, then s=infinity and sSafer=182.5.
  Therefore even though the average size of 8 data segments is unlikely to have been as small as MSS/8, d.cep cannot have been correct, because it would imply an average segment size greater than the MSS.

- if d.cep=2, dSafer.cep=10 and d.ceb=1460, then s=730 and sSafer=146.
  Therefore d.cep is safe enough, because the average size of 10 data segments is unlikely to have been as small as MSS/10.

- if d.cep=7, dSafer.cep=15 and d.ceb=10200, then s=1457 and sSafer=680.

Therefore d.cep is safe enough, because the average data segment size is more likely to have been just less than one MSS, rather than below MSS/2.

If pure ACKs were allowed to be ECN-capable, missing ACKs would be far less likely. However, because [RFC3168] currently precludes this, the above algorithm assumes that pure ACKs are not ECN-capable.

A.3. Example Algorithm to Estimate Marked Bytes from Marked Packets

If the AccECN Option is not available, the Data Sender can only decode CE-marking from the ACE field in packets. Every time an ACK arrives, to convert this into an estimate of CE-marked bytes, it needs an average of the segment size, s_ave. Then it can add or subtract s_ave from the value of d.ceb as the value of d.cep increments or decrements. Some possible ways to calculate s_ave are outlined below. The precise details will depend on why an estimate of marked bytes is needed.

The implementation could keep a record of the byte numbers of all the boundaries between packets in flight (including control packets), and recalculate s_ave on every ACK. However it would be simpler to merely maintain a counter packets_in_flight for the number of packets in flight (including control packets), which is reset once per RTT. Either way, it would estimate s_ave as:

```
s_ave ~= flightsize / packets_in_flight,
```

where flightsize is the variable that TCP already maintains for the number of bytes in flight. To avoid floating point arithmetic, it could right-bit-shift by lg(packets_in_flight), where lg() means log base 2.

An alternative would be to maintain an exponentially weighted moving average (EWMA) of the segment size:

```
s_ave = a * s + (1-a) * s_ave,
```

where a is the decay constant for the EWMA. However, then it is necessary to choose a good value for this constant, which ought to depend on the number of packets in flight. Also the decay constant needs to be power of two to avoid floating point arithmetic.

A.4. Example Algorithm to Beacon AccECN Options

Section 3.2.3.3 requires a Data Receiver to beacon a full-length AccECN Option at least 3 times per RTT. This could be implemented by maintaining a variable to store the number of ACKs (pure and data

ACKs) since a full AccECN Option was last sent and another for the approximate number of ACKs sent in the last round trip time:

```
   if (acks_since_full_last_sent > acks_in_round / BEACON_FREQ)
       send_full_AccECN_Option()
```

For optimized integer arithmetic, BEACON_FREQ = 4 could be used, rather than 3, so that the division could be implemented as an integer right bit-shift by lg(BEACON_FREQ).

In certain operating systems, it might be too complex to maintain acks_in_round. In others it might be possible by tagging each data segment in the retransmit buffer with the number of ACKs sent at the point that segment was sent. This would not work well if the Data Receiver was not sending data itself, in which case it might be necessary to beacon based on time instead, as follows:

```
   if ( time_now > time_last_option_sent + (RTT / BEACON_FREQ) )
       send_full_AccECN_Option()
```

This time-based approach does not work well when all the ACKs are sent early in each round trip, as is the case during slow-start. In this case few options will be sent (evtl. even less than 3 per RTT). However, when continuously sending data, data packets as well as ACKs will spread out equally over the RTT and sufficient ACKs with the AccECN option will be sent.

A.5. Example Algorithm to Count Not-ECT Bytes

A Data Sender in AccECN mode can infer the amount of TCP payload data arriving at the receiver marked Not-ECT from the difference between the amount of newly ACKed data and the sum of the bytes with the other three markings, d.ceb, d.e0b and d.e1b. Note that, because r.e0b is initialized to 1 and the other two counters are initialized to 0, the initial sum will be 1, which matches the initial offset of the TCP sequence number on completion of the 3WHS.

For this approach to be precise, it has to be assumed that spurious (unnecessary) retransmissions do not lead to double counting. This assumption is currently correct, given that RFC 3168 requires that the Data Sender marks retransmitted segments as Not-ECT. However, the converse is not true; necessary retransmissions will result in under-counting.

However, such precision is unlikely to be necessary. The only known use of a count of Not-ECT marked bytes is to test whether equipment on the path is clearing the ECN field (perhaps due to an out-dated attempt to clear, or bleach, what used to be the ToS field). To detect bleaching it will be sufficient to detect whether nearly all bytes arrive marked as Not-ECT. Therefore there should be no need to keep track of the details of retransmissions.

Appendix B. Rationale for Usage of TCP Header Flags

B.1. Three TCP Header Flags in the SYN-SYN/ACK Handshake

AccECN uses a rather unorthodox approach to negotiate the highest version TCP ECN feedback scheme that both ends support, as justified below. It follows from the original TCP ECN capability negotiation [RFC3168], in which the client set the 2 least significant of the original reserved flags in the TCP header, and fell back to no ECN support if the server responded with the 2 flags cleared, which had previously been the default.

ECN originally used header flags rather than a TCP option because it was considered more efficient to use a header flag for 1 bit of feedback per ACK, and this bit could be overloaded to indicate support for ECN during the handshake. During the development of ECN, 1 bit crept up to 2, in order to deliver the feedback reliably and to work round some broken hosts that reflected the reserved flags during the handshake.

In order to be backward compatible with RFC 3168, AccECN continues this approach, using the 3rd least significant TCP header flag that had previously been allocated for the ECN nonce (now historic). Then, whatever form of server an AccECN client encounters, the connection can fall back to the highest version of feedback protocol that both ends support, as explained in Section 3.1.

If AccECN had used the more orthodox approach of a TCP option, it would still have had to set the two ECN flags in the main TCP header, in order to be able to fall back to Classic RFC 3168 ECN, or to disable ECN support, without another round of negotiation. Then AccECN would also have had to handle all the different ways that servers currently respond to settings of the ECN flags in the main TCP header, including all the conflicting cases where a server might have said it supported one approach in the flags and another approach in the new TCP option. And AccECN would have had to deal with all the additional possibilities where a middlebox might have mangled the ECN flags, or removed the TCP option. Thus, usage of the 3rd reserved TCP header flag simplified the protocol.

The third flag was used in a way that could be distinguished from the ECN nonce, in case any nonce deployment was encountered. Previous usage of this flag for the ECN nonce was integrated into the original ECN negotiation. This further justified the 3rd flag's use for

AccECN, because a non-ECN usage of this flag would have had to use it as a separate single bit, rather than in combination with the other 2 ECN flags.

Indeed, having overloaded the original uses of these three flags for its handshake, AccECN overloads all three bits again as a 3-bit counter.

B.2. Four Codepoints in the SYN/ACK

Of the 8 possible codepoints that the 3 TCP header flags can indicate on the SYN/ACK, 4 already indicated earlier (or broken) versions of ECN support. In the early design of AccECN, an AccECN server could use only 2 of the 4 remaining codepoints. They both indicated AccECN support, but one fed back that the SYN had arrived marked as CE. Even though ECN support on a SYN is not yet on the standards track, the idea is for either end to act as a dumb reflector, so that future capabilities can be unilaterally deployed without requiring 2-ended deployment (justified in Section 2.5).

During traversal testing it was discovered that the ECN field in the SYN was mangled on a non-negligible proportion of paths. Therefore it was necessary to allow the SYN/ACK to feed all four IP/ECN codepoints that the SYN could arrive with back to the client. Without this, the client could not know whether to disable ECN for the connection due to mangling of the IP/ECN field (also explained in Section 2.5). This development consumed the remaining 2 codepoints on the SYN/ACK that had been reserved for future use by AccECN in earlier versions.

B.3. Space for Future Evolution

Despite availability of usable TCP header space being extremely scarce, the AccECN protocol has taken all possible steps to ensure that there is space to negotiate possible future variants of the protocol, either if the experiment proves that a variant of AccECN is required, or if a completely different ECN feedback approach is needed:

Future AccECN variants: When the AccECN capability is negotiated during TCP's 3WHS, the rows in Table 2 tagged as 'Nonce' and 'Broken' in the column for the capability of node B are unused by any current protocol in the RFC series. These could be used by TCP servers in future to indicate a variant of the AccECN protocol. In recent measurement studies in which the response of large numbers of servers to an AccECN SYN has been tested, e.g. [Mandalari18], a very small number of SYN/ACKs arrive with the pattern tagged as 'Nonce', and a small but more significant number arrive with the pattern tagged as ’Broken’. The ’Nonce’ pattern could be a sign that a few servers have implemented the ECN Nonce [RFC3540], which has now been reclassified as historic [RFC8311], or it could be the random result of some unknown middlebox behaviour. The greater prevalence of the ’Broken’ pattern suggests that some instances still exist of the broken code that reflects the reserved flags on the SYN.

The requirement not to reject unexpected initial values of the ACE counter (in the main TCP header) in the last para of Section 3.2.2.3 ensures that 3 unused codepoints on the ACK of the SYN/ACK, 6 unused values on the first SYN=0 data packet from the client and 7 unused values on the first SYN=0 data packet from the server could be used to declare future variants of the AccECN protocol. The word ’declare’ is used rather than ’negotiate’ because, at this late stage in the 3WHS, it would be too late for a negotiation between the endpoints to be completed. A similar requirement not to reject unexpected initial values in the TCP option (Section 3.2.3.2.4) is for the same purpose. If traversal of the TCP option were reliable, this would have enabled a far wider range of future variation of the whole AccECN protocol. Nonetheless, it could be used to reliably negotiate a wide range of variation in the semantics of the AccECN Option.

Future non-AccECN variants: Five codepoints out of the 8 possible in the 3 TCP header flags used by AccECN are unused on the initial SYN (in the order AE,CWR,ECE): 001, 010, 100, 101, 110. Section 3.1.3 ensures that the installed base of AccECN servers will all assume these are equivalent to AccECN negotiation with 111 on the SYN. These codepoints would not allow fall-back to Classic ECN support for a server that did not understand them, but this approach ensures they are available in future, perhaps for uses other than ECN alongside the AccECN scheme. All possible combinations of SYN/ACK could be used in response except either 000 or reflection of the same values sent on the SYN.

Of course, other ways could be resorted to in order to extend AccECN or ECN in future, although their traversal properties are likely to be inferior. They include a new TCP option; using the remaining reserved flags in the main TCP header (preferably extending the 3-bit combinations used by AccECN to 4-bit combinations, rather than burning one bit for just one state); a non-zero urgent pointer in combination with the URG flag cleared; or some other unexpected combination of fields yet to be invented.

Authors' Addresses

Bob Briscoe
Independent
UK

EMail: ietf@bobbriscoe.net
URI: http://bobbriscoe.net/

Mirja Kuehlewind
Ericsson
Germany

EMail: ietf@kuehlewind.net

Richard Scheffenegger
NetApp
Vienna
Austria

EMail: Richard.Scheffenegger@netapp.com

Network Working Group
Internet-Draft
Obsoletes: 5562 (if approved)
Intended status: Experimental
Expires: May 1, 2021

M. Bagnulo
UC3M
B. Briscoe
Independent

October 28, 2020

ECN++: Adding Explicit Congestion Notification (ECN) to TCP Control Packets
draft-ietf-tcpm-generalized-ecn-06

Abstract

This document describes an experimental modification to ECN when used with TCP. It allows the use of ECN on the following TCP packets: SYNs, pure ACKs, Window probes, FINs, RSTs and retransmissions.

Status of This Memo

This Internet-Draft will expire on May 1, 2021.

Copyright Notice

Table of Contents

1. Introduction

RFC 3168 [RFC3168] specifies support of Explicit Congestion Notification (ECN) in IP (v4 and v6). By using the ECN capability, network elements (e.g. routers, switches) performing Active Queue Management (AQM) can use ECN marks instead of packet drops to signal congestion to the endpoints of a communication. This results in lower packet loss and increased performance. RFC 3168 also specifies support for ECN in TCP, but solely on data packets. For various reasons it precludes the use of ECN on TCP control packets (TCP SYN, TCP SYN-ACK, pure ACKs, Window probes) and on retransmitted packets. RFC 3168 is silent about the use of ECN on RST and FIN packets. RFC 5562 [RFC5562] is an experimental modification to ECN that enables ECN support for TCP SYN-ACK packets.

This document defines an experimental modification to ECN [RFC3168] that shall be called ECN++. It enables ECN support on all the aforementioned types of TCP packet. The mechanisms proposed in this document have been defined conservatively and with safety in mind, possibly in some cases at the expense of performance.

ECN++ uses a sender-only deployment model. It works whether the two ends of the TCP connection use classic ECN feedback [RFC3168] or experimental Accurate ECN feedback (AccECN

[I-D.ietf-tcpm-accurate-ecn]), the two ECN feedback mechanisms for TCP being standardized at the time of writing.

Using ECN on initial SYN packets provides significant benefits, as we describe in the next subsection. However, only AccECN provides a way to feed back whether the SYN was CE marked, and RFC 3168 does not. Therefore, implementers of ECN++ are RECOMMENDED to also implement AccECN. Conversely, if AccECN (or an equivalent safety mechanism) is not implemented with ECN++, this specification rules out ECN on the SYN.

ECN++ is designed for compatibility with a number of latency improvements to TCP such as TCP Fast Open (TFO [RFC7413]), initial window of 10 SMSS (IW10 [RFC6928]) and Low latency Low Loss Scalable Transport (L4S [I-D.ietf-tsvwg-l4s-arch]), but they can all be implemented and deployed independently. [RFC8311] is a standards track procedural device that relaxes requirements in RFC 3168 and other standards track RFCs that would otherwise preclude the experimental modifications needed for ECN++ and other ECN experiments.

1.1. Motivation

The absence of ECN support on TCP control packets and retransmissions has a potential harmful effect. In any ECN deployment, non-ECN-capable packets suffer a penalty when they traverse a congested bottleneck. For instance, with a drop probability of 1%, 1% of connection attempts suffer a timeout of about 1 second before the SYN is retransmitted, which is highly detrimental to the performance of short flows. TCP control packets, particularly TCP SYNs and SYN-ACKs, are important for performance, so dropping them is best avoided.

Not using ECN on control packets can be particularly detrimental to performance in environments where the ECN marking level is high. For example, [judd-nsdi] shows that in a controlled private data centre (DC) environment where ECN is used (in conjunction with DCTCP [RFC8257]), the probability of being able to establish a new connection using a non-ECN SYN packet drops to close to zero even when there are only 16 ongoing TCP flows transmitting at full speed. The issue is that DCTCP exhibits a much more aggressive response to packet marking (which is why it is only applicable in controlled environments). This leads to a high marking probability for ECN-capable packets, and in turn a high drop probability for non-ECN packets. Therefore non-ECN SYNs are dropped aggressively, rendering it nearly impossible to establish a new connection in the presence of even mild traffic load.

Finally, there are ongoing experimental efforts to promote the adoption of a slightly modified variant of DCTCP (and similar congestion controls) over the Internet to achieve low latency, low loss and scalable throughput (L4S) for all communications [I-D.ietf-tsvwg-l4s-arch]. In such an approach, L4S packets identify themselves using an ECN codepoint [I-D.ietf-tsvwg-ecn-l4s-id]. With L4S, preventing TCP control packets from obtaining the benefits of ECN would not only expose them to the prevailing level of congestion loss, but it would also classify them into a different queue. Then only L4S data packets would be classified into the L4S queue that is expected to have lower latency, while the packets controlling and retransmitting these data packets would still get stuck behind the queue induced by non-L4S-enabled TCP traffic.

1.2. Experiment Goals

The goal of the experimental modifications defined in this document is to allow the use of ECN on all TCP packets. Experiments are expected in the public Internet as well as in controlled environments to understand the following issues:

- How SYNs, Window probes, pure ACKs, FINs, RSTs and retransmissions that carry the ECT(0), ECT(1) or CE codepoints are processed by the TCP endpoints and the network (including routers, firewalls and other middleboxes). In particular we would like to learn if these packets are frequently blocked or if these packets are usually forwarded and processed.

- The scale of deployment of the different flavours of ECN, including [RFC3168], [RFC5562], [RFC3540] and [I-D.ietf-tcpm-accurate-ecn].

- How much the performance of TCP communications is improved by allowing ECN marking of each packet type.

- To identify any issues (including security issues) raised by enabling ECN marking of these packets.

- To conduct the specific experiments identified in the text by the strings "EXPERIMENTATION NEEDED" or "MEASUREMENTS NEEDED".

The data gathered through the experiments described in this document, particularly under the first 2 bullets above, will help in the redesign of the final mechanism (if needed) for adding ECN support to the different packet types considered in this document.

Success criteria: The experiment will be a success if we obtain enough data to have a clearer view of the deployability and benefits of enabling ECN on all TCP packets, as well as any issues. If the results of the experiment show that it is feasible to deploy such changes; that there are gains to be achieved through the changes described in this specification; and that no other major issues may interfere with the deployment of the proposed changes; then it would be reasonable to adopt the proposed changes in a standards track specification that would update RFC 3168.

1.3. Document Structure

The remainder of this document is structured as follows. In Section 2, we present the terminology used in the rest of the document. In Section 3, we specify the modifications to provide ECN support to TCP SYNs, pure ACKs, Window probes, FINs, RSTs and retransmissions. We describe both the network behaviour and the endpoint behaviour. Section 5 discusses variations of the specification that will be necessary to interwork with a number of popular variants or derivatives of TCP. RFC 3168 provides a number of specific reasons why ECN support is not appropriate for each packet type. In Section 4, we revisit each of these arguments for each packet type to justify why it is reasonable to conduct this experiment.

2. Terminology

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, NOT RECOMMENDED, MAY, and OPTIONAL in this document, are to be interpreted as described in BCP 14 [RFC2119] when and only when they appear in all capitals [RFC8174].

Pure ACK: A TCP segment with the ACK flag set and no data payload.

SYN: A TCP segment with the SYN (synchronize) flag set.

Window probe: Defined in [RFC0793], a window probe is a TCP segment with only one byte of data sent to learn if the receive window is still zero.

FIN: A TCP segment with the FIN (finish) flag set.

RST: A TCP segment with the RST (reset) flag set.

Retransmission: A TCP segment that has been retransmitted by the TCP sender.

TCP client: The initiating end of a TCP connection. Also called the initiator.

TCP server: The responding end of a TCP connection. Also called the responder.

ECT: ECN-Capable Transport. One of the two codepoints ECT(0) or ECT(1) in the ECN field [RFC3168] of the IP header (v4 or v6). An ECN-capable sender sets one of these to indicate that both transport end-points support ECN. When this specification says the sender sets an ECT codepoint, by default it means ECT(0). Optionally, it could mean ECT(1), which is in the process of being redefined for use by L4S experiments [RFC8311] [I-D.ietf-tsvwg-ecn-l4s-id].

Not-ECT: The ECN codepoint set by senders that indicates that the transport is not ECN-capable.

CE: Congestion Experienced. The ECN codepoint that an intermediate node sets to indicate congestion [RFC3168]. A node sets an increasing proportion of ECT packets to CE as the level of congestion increases.

3. Specification

The experimental ECN++ changes to the specification of TCP over ECN [RFC3168] defined here primarily alter the behaviour of the sending host for each half-connection. However, there are subsections for forwarding elements and receivers below, which recommend that they accept the new packets - they should do already, but might not. This will allow implementers to check the receive side code while they are altering the send-side code. All changes can be deployed at each end-point independently of others and independent of any network behaviour.

The feedback behaviour at the receiver depends on whether classic ECN TCP feedback [RFC3168] or Accurate ECN (AccECN) TCP feedback [I-D.ietf-tcpm-accurate-ecn] has been negotiated. Nonetheless, neither receiver feedback behaviour is altered by the present specification.

3.1. Network (e.g. Firewall) Behaviour

Previously the specification of ECN for TCP [RFC3168] required the sender to set not-ECT on TCP control packets and retransmissions. Some readers of RFC 3168 might have erroneously interpreted this as a requirement for firewalls, intrusion detection systems, etc. to check and enforce this behaviour. Section 4.3 of [RFC8311] updates RFC 3168 to remove this ambiguity. It requires firewalls or any intermediate nodes not to treat certain types of ECN-capable TCP segment differently (except potentially in one attack scenario). This is likely to only involve a firewall rule change in a fraction of cases (at most 0.4% of paths according to the tests reported in Section 4.2.2).

In case a TCP sender encounters a middlebox blocking ECT on certain TCP segments, the specification below includes behaviour to fall back to non-ECN. However, this loses the benefit of ECN on control packets. So operators are RECOMMENDED to alter their firewall rules to comply with the requirement referred to above (section 4.3 of [RFC8311]).

3.2. Sender Behaviour

For each type of control packet or retransmission, the following sections detail changes to the sender's behaviour in two respects: i) whether it sets ECT; and ii) its response to congestion feedback. Table 1 summarises these two behaviours for each type of packet, but the relevant subsection below should be referred to for the detailed behaviour. The subsection on the SYN is more complex than the others, because it has to include fall-back behaviour if the ECT packet appears not to have got through, and caching of the outcome to detect persistent failures.

TCP packet type	ECN field if AccECN f/b negotiated*	ECN field if RFC3168 f/b negotiated*	Congestion Response
SYN	ECT	not-ECT	If AccECN, reduce IW
SYN-ACK	ECT	ECT	Reduce IW
Pure ACK	ECT	not-ECT	If AccECN, usual cwnd response and optionally [RFC5690]
W Probe	ECT	ECT	Usual cwnd response
FIN	ECT	ECT	None or optionally [RFC5690]
RST	ECT	ECT	N/A
Re-XMT	ECT	ECT	Usual cwnd response

Window probe and retransmission are abbreviated to W Probe an Re-XMT.
* For a SYN, "negotiated" means "requested".

Table 1: Summary of sender behaviour. In each case the relevant section below should be referred to for the detailed behaviour

It can be seen that we recommend against the sender setting ECT on the SYN if it is not requesting AccECN feedback. Therefore it is RECOMMENDED that the experimental AccECN specification [I-D.ietf-tcpm-accurate-ecn] is implemented, along with the ECN++ experiment, because it is expected that ECT on the SYN will give the most significant performance gain, particularly for short flows.

Nonetheless, this specification also caters for the case where an ECN++ TCP sender is not using AccECN. This could be because it does not support AccECN or because the other end of the TCP connection does not (AccECN can only be used for a connection if both ends support it).

3.2.1. SYN (Send)

3.2.1.1. Setting ECT on the SYN

With classic [RFC3168] ECN feedback, the SYN was not expected to be ECN-capable, so the flag provided to feed back congestion was put to another use (it is used in combination with other flags to indicate that the responder supports ECN). In contrast, Accurate ECN (AccECN) feedback [I-D.ietf-tcpm-accurate-ecn] provides a codepoint in the SYN-ACK for the responder to feed back whether the SYN arrived marked CE. Therefore the setting of the IP/ECN field on the SYN is specified separately for each case in the following two subsections.

3.2.1.1.1. ECN++ TCP Client also Supports AccECN

For the ECN++ experiment, if the SYN is requesting AccECN feedback, the TCP sender will also set ECT on the SYN. It can ignore the prohibition in section 6.1.1 of RFC 3168 against setting ECT on such a SYN, as per Section 4.3 of [RFC8311].

3.2.1.1.2. ECN++ TCP Client does not Support AccECN

If the SYN sent by a TCP initiator does not attempt to negotiate Accurate ECN feedback, or does not use an equivalent safety mechanism, it MUST still comply with RFC 3168, which says that a TCP initiator "MUST NOT set ECT on a SYN".

The only envisaged examples of "equivalent safety mechanisms" are: a) some future TCP ECN feedback protocol, perhaps evolved from AccECN, that feeds back CE marking on a SYN; b) setting the initial window to 1 SMSS. IW=1 is NOT RECOMMENDED because it could degrade performance, but might be appropriate for certain lightweight TCP implementations.

See Section 4.2 for discussion and rationale.

If the TCP initiator does not set ECT on the SYN, the rest of Section 3.2.1 does not apply.

3.2.1.2. Caching where to use ECT on SYNs

This subsection only applies if the ECN++ TCP client set ECTs on the SYN and supports AccECN.

Until AccECN servers become widely deployed, a TCP initiator that sets ECT on a SYN (which typically implies the same SYN also requests AccECN, as above) SHOULD also maintain a cache entry per server to record servers that it is not worth sending an ECT SYN to, e.g. because they do not support AccECN and therefore have no logic for congestion markings on the SYN. Mobile hosts MAY maintain a cache entry per access network to record 'non-ECT SYN' entries against proxies (see Section 4.2.3). This cache can be implemented as part of the shared state across multiple TCP connections, following [RFC2140].

Subsequently the initiator will not set ECT on a SYN to such a server or proxy, but it can still always request AccECN support (because the response will state any earlier stage of ECN evolution that the server supports with no performance penalty). If a server subsequently upgrades to support AccECN, the initiator will discover this as soon as it next connects, then it can remove the server from its cache and subsequently always set ECT for that server.

The client can limit the size of its cache of 'non-ECT SYN' servers. Then, while AccECN is not widely deployed, it will only cache the 'non-ECT SYN' servers that are most used and most recently used by the client. As the client accesses servers that have been expelled from its cache, it will simply use ECT on the SYN by default.

Servers that do not support ECN as a whole do not need to be recorded separately from non-support of AccECN because the response to a request for AccECN immediately states which stage in the evolution of ECN the server supports (AccECN [I-D.ietf-tcpm-accurate-ecn], classic ECN [RFC3168] or no ECN).

The above strategy is named "optimistic ECT and cache failures". It is believed to be sufficient based on three measurement studies and assumptions detailed in Section 4.2.3. However, Section 4.2.3 gives two other strategies and the choice between them depends on the implementer's goals and the deployment prevalence of ECN variants in the network and on servers, not to mention the prevalence of some significant bugs.

If the initiator times out without seeing a SYN-ACK, it will separately cache this fact (see fall-back in Section 3.2.1.4 for details).

3.2.1.3. SYN Congestion Response

As explained above, this subsection only applies if the ECN++ TCP client sets ECT on the initial SYN.

If the SYN-ACK returned to the TCP initiator confirms that the server supports AccECN, it will also be able to indicate whether or not the SYN was CE-marked. If the SYN was CE-marked, and if the initial window is greater than 1 MSS, then, the initiator MUST reduce its Initial Window (IW) and SHOULD reduce it to 1 SMSS (sender maximum segment size). The rationale is the same as that for the response to CE on a SYN-ACK (Section 4.3.2).

If the initiator has set ECT on the SYN and if the SYN-ACK shows that the server does not support feedback of a CE on the SYN (e.g. it does not support AccECN) and if the initial congestion window of the initiator is greater than 1 MSS, then the TCP initiator MUST conservatively reduce its Initial Window and SHOULD reduce it to 1 SMSS. A reduction to greater than 1 SMSS MAY be appropriate (see Section 4.2.1). Conservatism is necessary because the SYN-ACK cannot show whether the SYN was CE-marked.

If the TCP initiator (host A) receives a SYN from the remote end (host B) after it has sent a SYN to B, it indicates the (unusual) case of a simultaneous open. Host A will respond with a SYN-ACK. Host A will probably then receive a SYN-ACK in response to its own SYN, after which it can follow the appropriate one of the two paragraphs above.

In all the above cases, the initiator does not have to back off its retransmission timer as it would in response to a timeout following no response to its SYN [RFC6298], because both the SYN and the SYN-ACK have been successfully delivered through the network. Also, the initiator does not need to exit slow start or reduce ssthresh, which is not even required when a SYN is lost [RFC5681].

If an initial window of more than 3 segments is implemented (e.g. IW10 [RFC6928]), Section 5 gives additional recommendations.

3.2.1.4. Fall-Back Following No Response to an ECT SYN

As explained above, this subsection only applies if the ECN++ TCP client also sets ECT on the initial SYN.

An ECT SYN might be lost due to an over-zealous path element (or server) blocking ECT packets that do not conform to RFC 3168. Some evidence of this was found in a 2014 study [ecn-pam], but in a more recent study using 2017 data [Mandalari18] extensive measurements found no case where ECT on TCP control packets was treated any differently from ECT on TCP data packets. Loss is commonplace for numerous other reasons, e.g. congestion loss at a non-ECN queue on the forward or reverse path, transmission errors, etc. Alternatively, the cause of the loss might be the associated attempt to negotiate AccECN, or possibly other unrelated options on the SYN.

Therefore, if the timer expires after the TCP initiator has sent the first ECT SYN, it SHOULD make one more attempt to retransmit the SYN with ECT set (backing off the timer as usual). If the retransmission timer expires again, it SHOULD retransmit the SYN with the not-ECT codepoint in the IP header, to expedite connection set-up. If other experimental fields or options were on the SYN, it will also be necessary to follow their specifications for fall-back too. It would make sense to coordinate all the strategies for fall-back in order to isolate the specific cause of the problem.

If the TCP initiator is caching failed connection attempts, it SHOULD NOT give up using ECT on the first SYN of subsequent connection attempts until it is clear that a blockage persistently and specifically affects ECT on SYNs. This is because loss is so commonplace for other reasons. Even if it does eventually decide to give up setting ECT on the SYN, it will probably not need to give up on AccECN on the SYN. In any case, if a cache is used, it SHOULD be arranged to expire so that the initiator will infrequently attempt to check whether the problem has been resolved.

Other fall-back strategies MAY be adopted where applicable (see Section 4.2.2 for suggestions, and the conditions under which they would apply).

3.2.2. SYN-ACK (Send)

3.2.2.1. Setting ECT on the SYN-ACK

For the ECN++ experiment, the TCP implementation will set ECT on SYN-ACKs. It can ignore the requirement in section 6.1.1 of RFC 3168 to set not-ECT on a SYN-ACK, as per Section 4.3 of [RFC8311].

3.2.2.2. SYN-ACK Congestion Response

A host that sets ECT on SYN-ACKs MUST reduce its initial window in response to any congestion feedback, whether using classic ECN or AccECN (see Section 4.3.1). It SHOULD reduce it to 1 SMSS. This is different to the behaviour specified in an earlier experiment that set ECT on the SYN-ACK [RFC5562]. This is justified in Section 4.3.2.

The responder does not have to back off its retransmission timer because the ECN feedback proves that the network is delivering packets successfully and is not severely overloaded. Also the responder does not have to leave slow start or reduce ssthresh, which is not even required when a SYN-ACK has been lost.

The congestion response to CE-marking on a SYN-ACK for a server that implements either the TCP Fast Open experiment (TFO [RFC7413]) or experimentation with an initial window of more than 3 segments (e.g. IW10 [RFC6928]) is discussed in Section 5.

3.2.2.3. Fall-Back Following No Response to an ECT SYN-ACK

After the responder sends a SYN-ACK with ECT set, if its retransmission timer expires it SHOULD retransmit one more SYN-ACK with ECT set (and back-off its timer as usual). If the timer expires again, it SHOULD retransmit the SYN-ACK with not-ECT in the IP header. If other experimental fields or options were on the initial SYN-ACK, it will also be necessary to follow their specifications for fall-back. It would make sense to co-ordinate all the strategies for fall-back in order to isolate the specific cause of the problem.

This fall-back strategy attempts to use ECT one more time than the strategy for ECT SYN-ACKs in [RFC5562] (which is made obsolete, being superseded by the present specification). Other fall-back strategies MAY be adopted if found to be more effective, e.g. fall-back to not-ECT on the first retransmission attempt.

The server MAY cache failed connection attempts, e.g. per client access network. A client-based alternative to caching at the server is given in Section 4.3.3. If the TCP server is caching failed connection attempts, it SHOULD NOT give up using ECT on the first SYN-ACK of subsequent connection attempts until it is clear that the blockage persistently and specifically affects ECT on SYN-ACKs. This is because loss is so commonplace for other reasons (see Section 3.2.1.4). If a cache is used, it SHOULD be arranged to expire so that the server will infrequently attempt to check whether the problem has been resolved.

3.2.3. Pure ACK (Send)

A Pure ACK is an ACK packet that does not carry data, which includes the Pure ACK at the end of TCP’s 3-way handshake.

For the ECN++ experiment, whether a TCP implementation sets ECT on a Pure ACK depends on whether or not Accurate ECN TCP feedback [I-D.ietf-tcpm-accurate-ecn] has been successfully negotiated for a particular TCP connection, as specified in the following two subsections.

3.2.3.1. Pure ACK without AccECN Feedback

If AccECN has not been successfully negotiated for a connection, ECT MUST NOT be set on Pure ACKs by either end.

3.2.3.2. Pure ACK with AccECN Feedback

For the ECN++ experiment, if AccECN has been successfully negotiated, either end of the connection will set ECT on Pure ACKs. They can ignore the requirement in section 6.1.4 of RFC 3168 to set not-ECT on a pure ACK, as per Section 4.3 of [RFC8311].

> MEASUREMENTS NEEDED: Measurements are needed to learn how the deployed base of network elements and RFC 3168 servers react to pure ACKs marked with the ECT(0)/ECT(1)/CE codepoints, i.e. whether they are dropped, codepoint cleared or processed and the congestion indication fed back on a subsequent packet.

See Section 3.3.3 for the implications if a host receives a CE-marked Pure ACK.

3.2.3.2.1. Pure ACK Congestion Response

As explained above, this subsection only applies if AccECN has been successfully negotiated for the TCP connection.

A host that sets ECT on pure ACKs SHOULD respond to the congestion signal resulting from pure ACKs being marked with the CE codepoint. The specific response will need to be defined as an update to each congestion control specification. Possible responses to congestion feedback include reducing the congestion window (CWND) and/or regulating the pure ACK rate (see Section 4.4.1.1).

Note that, in comparison, TCP Congestion Control [RFC5681] does not require a TCP to detect or respond to loss of pure ACKs at all; it requires no reduction in congestion window or ACK rate.

3.2.4. Window Probe (Send)

For the ECN++ experiment, the TCP sender will set ECT on window probes. It can ignore the prohibition in section 6.1.6 of RFC 3168 against setting ECT on a window probe, as per Section 4.3 of [RFC8311].

A window probe contains a single octet, so it is no different from a regular TCP data segment. Therefore a TCP receiver will feed back any CE marking on a window probe as normal (either using classic ECN feedback or AccECN feedback). The sender of the probe will then reduce its congestion window as normal.

A receive window of zero indicates that the application is not consuming data fast enough and does not imply anything about network congestion. Once the receive window opens, the congestion window might become the limiting factor, so it is correct that CE-marked probes reduce the congestion window. This complements cwnd validation [RFC7661], which reduces cwnd as more time elapses without having used available capacity. However, CE-marking on window probes does not reduce the rate of the probes themselves. This is unlikely to present a problem, given the duration between window probes doubles [RFC1122] as long as the receiver is advertising a zero window (currently minimum 1 second, maximum at least 1 minute [RFC6298]).

> MEASUREMENTS NEEDED: Measurements are needed to learn how the deployed base of network elements and servers react to Window probes marked with the ECT(0)/ECT(1)/CE codepoints, i.e. whether they are dropped, codepoint cleared or processed.

3.2.5. FIN (Send)

A TCP implementation can set ECT on a FIN.

See Section 3.3.4 for the implications if a host receives a CE-marked FIN.

A congestion response to a CE-marking on a FIN is not required.

After sending a FIN, the endpoint will not send any more data in the connection. Therefore, even if the FIN-ACK indicates that the FIN was CE-marked (whether using classic or AccECN feedback), reducing the congestion window will not affect anything.

After sending a FIN, a host might send one or more pure ACKs. If it is using one of the techniques in Section 3.2.3 to regulate the delayed ACK ratio for pure ACKs, it could equally be applied after a FIN. But this is not required.

> MEASUREMENTS NEEDED: Measurements are needed to learn how the deployed base of network elements and servers react to FIN packets marked with the ECT(0)/ECT(1)/CE codepoints, i.e. whether they are dropped, codepoint cleared or processed.

3.2.6. RST (Send)

A TCP implementation can set ECT on a RST.

See Section 3.3.5 for the implications if a host receives a CE-marked RST.

A congestion response to a CE-marking on a RST is not required (and actually not possible).

MEASUREMENTS NEEDED: Measurements are needed to learn how the deployed base of network elements and servers react to RST packets marked with the ECT(0)/ECT(1)/CE codepoints, i.e. whether they are dropped, codepoint cleared or processed.

3.2.7. Retransmissions (Send)

For the ECN++ experiment, the TCP sender will set ECT on retransmitted segments. It can ignore the prohibition in section 6.1.5 of RFC 3168 against setting ECT on retransmissions, as per Section 4.3 of [RFC8311].

See Section 3.3.6 for the implications if a host receives a CE-marked retransmission.

If the TCP sender receives feedback that a retransmitted packet was CE-marked, it will react as it would to any feedback of CE-marking on a data packet.

MEASUREMENTS NEEDED: Measurements are needed to learn how the deployed base of network elements and servers react to retransmissions marked with the ECT(0)/ECT(1)/CE codepoints, i.e. whether they are dropped, codepoint cleared or processed.

3.2.8. General Fall-back for any Control Packet or Retransmission

Extensive measurements in fixed and mobile networks [Mandalari18] have found no evidence of blockages due to ECT being set on any type of TCP control packet.

In case traversal problems arise in future, fall-back measures have been specified above, but only for the cases where ECT on the initial packet of a half-connection (SYN or SYN-ACK) is persistently failing to get through.

Fall-back measures for blockage of ECT on other TCP control packets MAY be implemented. However they are not specified here given the lack of any evidence they will be needed. Section 4.9 justifies this advice in more detail.

3.3. Receiver Behaviour

The present ECN++ specification primarily concerns the behaviour for sending TCP control packets or retransmissions. Below are a few changes to the receive side of an implementation that are recommended while updating its send side. Nonetheless, where deployment is concerned, ECN++ is still a sender-only deployment, because it does not depend on receivers complying with any of these recommendations.

3.3.1. Receiver Behaviour for Any TCP Control Packet or Retransmission

RFC8311 is a standards track update to RFC 3168 in order to (amongst other things) "...allow the use of ECT codepoints on SYN packets, pure acknowledgement packets, window probe packets, and retransmissions of packets..., provided that the changes from RFC 3168 are documented in an Experimental RFC in the IETF document stream."

Section 4.3 of RFC 8311 amends every statement in RFC 3168 that precludes the use of ECT on control packets and retransmissions to add "unless otherwise specified by an Experimental RFC in the IETF document stream". The present specification is such an Experimental RFC. Therefore, In order for this experiment to be useful, the following requirements follow from RFC8311:

o Any TCP implementation SHOULD accept receipt of any valid TCP control packet or retransmission irrespective of its IP/ECN field. If any existing implementation does not, it SHOULD be updated to do so.

o A TCP implementation taking part in the experiments proposed here MUST accept receipt of any valid TCP control packet or retransmission irrespective of its IP/ECN field.

These measures are derived from the robustness principle of "... be liberal in what you accept from others", in order to ensure compatibility with any future protocol changes that allow ECT on any TCP packet.

3.3.2. SYN (Receive)

RFC 3168 negotiates the use of ECN for the connection end-to-end using the ECN flags in the TCP header. When RFC3168 says that "A host MUST NOT set ECT on SYN ... packets." it is silent as to what a TCP server ought to do if it receives a SYN packet with a non-zero IP/ECN field.

As the time of the writing, some implementations of TCP servers (see Section 4.2.2.2) assume that, if a host receives a SYN with a non-zero IP/ECN field, it must be due to network mangling, and they disable ECN for the rest of the connection. Section 4.2.2.2 also finds that this type of network mangling seems to be virtually non-existent so it would be preferable to report any such mangling so it can be fixed.

For the avoidance of doubt, the normative statements for all TCP control packets in Section 3.3.1 are interpreted for the case when a SYN is received as follows:

- Any TCP server implementation SHOULD accept receipt of a valid SYN that requests ECN support for the connection, irrespective of the IP/ECN field of the SYN. If any existing implementation does not, it SHOULD be updated to do so.

- A TCP implementation taking part in the ECN++ experiment MUST accept receipt of a valid SYN, irrespective of its IP/ECN field.

- If the SYN is CE-marked and the server has no logic to feed back a CE mark on a SYN-ACK (e.g. it does not support AccECN), it has to ignore the CE-mark (the client detects this case and behaves conservatively in mitigation - see Section 3.2.1.3).

3.3.3. Pure ACK (Receive)

For the avoidance of doubt, the normative statements for all TCP control packets in Section 3.3.1 are interpreted for the case when a Pure ACK is received as follows:

- Any TCP implementation SHOULD accept receipt of a pure ACK with a non-zero ECN field, despite current RFCs precluding the sending of such packets.

- A TCP implementation taking part in the ECN++ experiment MUST accept receipt of a pure ACK with a non-zero ECN field.

The question of whether and how the receiver of pure ACKs is required to feed back any CE marks on them is outside the scope of the present specification because it is a matter for the relevant feedback specification ([RFC3168] or [I-D.ietf-tcpm-accurate-ecn]). AccECN feedback is required to count CE marking of any control packet including pure ACKs. Whereas RFC 3168 is silent on this point, so feedback of CE-markings might be implementation specific (see Section 4.4.1.1).

3.3.4. FIN (Receive)

The TCP data receiver MUST ignore the CE codepoint on incoming FINs that fail any validity check. The validity check in section 5.2 of [RFC5961] is RECOMMENDED.

3.3.5. RST (Receive)

The "challenge ACK" approach to checking the validity of RSTs (section 3.2 of [RFC5961] is RECOMMENDED at the data receiver.

3.3.6. Retransmissions (Receive)

The TCP data receiver MUST ignore the CE codepoint on incoming segments that fail any validity check. The validity check in section 5.2 of [RFC5961] is RECOMMENDED. This will effectively mitigate an attack that uses spoofed data packets to fool the receiver into feeding back spoofed congestion indications to the sender, which in turn would be fooled into continually reducing its congestion window.

4. Rationale

This section is informative, not normative. It presents counter-arguments against the justifications in the RFC series for disabling ECN on TCP control segments and retransmissions. It also gives rationale for why ECT is safe on control segments that have not, so far, been mentioned in the RFC series. First it addresses over-arching arguments used for most packet types, then it addresses the specific arguments for each packet type in turn.

4.1. The Reliability Argument

Section 5.2 of RFC 3168 states:

> "To ensure the reliable delivery of the congestion indication of the CE codepoint, an ECT codepoint MUST NOT be set in a packet unless the loss of that packet [at a subsequent node] in the network would be detected by the end nodes and interpreted as an indication of congestion."

We believe this argument is misplaced. TCP does not deliver most control packets reliably. So it is more important to allow control packets to be ECN-capable, which greatly improves reliable delivery of the control packets themselves (see motivation in Section 1.1). ECN also improves the reliability and latency of delivery of any congestion notification on control packets, particularly because TCP does not detect the loss of most types of control packet anyway. Both these points outweigh by far the concern that a CE marking applied to a control packet by one node might subsequently be dropped by another node.

The principle to determine whether a packet can be ECN-capable ought to be "do no extra harm", meaning that the reliability of a congestion signal’s delivery ought to be no worse with ECN than without. In particular, setting the CE codepoint on the very same packet that would otherwise have been dropped fulfills this criterion, since either the packet is delivered and the CE signal is delivered to the endpoint, or the packet is dropped and the original congestion signal (packet loss) is delivered to the endpoint.

The concern about a CE marking being dropped at a subsequent node might be motivated by the idea that ECN-marking a packet at the first node does not remove the packet, so it could go on to worsen congestion at a subsequent node. However, it is not useful to reason about congestion by considering single packets. The departure rate from the first node will generally be the same (fully utilized) with or without ECN, so this argument does not apply.

4.2. SYNs

RFC 5562 presents two arguments against ECT marking of SYN packets (quoted verbatim):

> "First, when the TCP SYN packet is sent, there are no guarantees that the other TCP endpoint (node B in Figure 2) is ECN-Capable, or that it would be able to understand and react if the ECN CE codepoint was set by a congested router.
>
> Second, the ECN-Capable codepoint in TCP SYN packets could be misused by malicious clients to "improve" the well-known TCP SYN attack. By setting an ECN-Capable codepoint in TCP SYN packets, a malicious host might be able to inject a large number of TCP SYN packets through a potentially congested ECN-enabled router, congesting it even further."

The first point actually describes two subtly different issues. So below three arguments are countered in turn.

4.2.1. Argument 1a: Unrecognized CE on the SYN

This argument certainly applied at the time RFC 5562 was written, when no ECN responder mechanism had any logic to recognize a CE marking on a SYN and, even if logic were added, there was no field in the SYN-ACK to feed it back. The problem was that, during the 3WHS, the flag in the TCP header for ECN feedback (called Echo Congestion Experienced) had been overloaded to negotiate the use of ECN itself.

The accurate ECN (AccECN) protocol [I-D.ietf-tcpm-accurate-ecn] has since been designed to solve this problem. Two features are important here:

1. An AccECN server uses the 3 'ECN' bits in the TCP header of the SYN-ACK to respond to the client. 4 of the possible 8 codepoints provide enough space for the server to feed back which of the 4 IP/ECN codepoints was on the incoming SYN (including CE of course).

2. If any of these 4 codepoints are in the SYN-ACK, it confirms that the server supports AccECN and, if another codepoint is returned, it confirms that the server doesn't support AccECN.

This still does not seem to allow a client to set ECT on a SYN, it only finds out whether the server would have supported it afterwards. The trick the client uses for ECN++ is to set ECT on the SYN optimistically then, if the SYN-ACK reveals that the server wouldn't have understood CE on the SYN, the client responds conservatively as if the SYN was marked with CE.

The recommended conservative congestion response is to reduce the initial window, which does not affect the performance of very popular protocols such as HTTP, since it is extremely rare for an HTTP client to send more than one packet as its initial request anyway (for data on HTTP/1 & HTTP/2 request sizes see Fig 3 in [Manzoor17]). Any clients that do frequently use a larger initial window for their first message to the server can cache which servers will not understand ECT on a SYN (see Section 4.2.3 below). If caching is not practical, such clients could reduce the initial window to say IW2 or IW3.

> EXPERIMENTATION NEEDED: Experiments will be needed to determine any better strategy for reducing IW in response to congestion on a SYN, when the server does not support congestion feedback on the SYN-ACK (whether cached or discovered explicitly).

4.2.2. Argument 1b: ECT Considered Invalid on the SYN

Given, until now, ECT-marked SYN packets have been prohibited, it cannot be assumed they will be accepted, by TCP middleboxes or servers.

4.2.2.1. ECT on SYN Considered Invalid by Middleboxes

According to a study using 2014 data [ecn-pam] from a limited range of fixed vantage points, for the top 1M Alexa web sites, adding the ECN capability to SYNs was increasing connection establishment failures by about 0.4%.

From a wider range of fixed and mobile vantage points, a more recent study in Jan-May 2017 [Mandalari18] found no occurrences of blocking of ECT on SYNs. However, in more than half the mobile networks tested it found wiping of the ECN codepoint at the first hop.

> MEASUREMENTS NEEDED: As wiping at the first hop is remedied, measurements will be needed to check whether SYNs with ECT are sometimes blocked deeper into the path.

Silent failures introduce a retransmission timeout delay (default 1 second) at the initiator before it attempts any fall back strategy (whereas explicit RSTs can be dealt with immediately). Ironically, making SYNs ECN-capable is intended to avoid the timeout when a SYN is lost due to congestion. Fortunately, if there is any discard of ECN-capable SYNs due to policy, it will occur predictably, not randomly like congestion. So the initiator should be able to avoid it by caching those sites that do not support ECN-capable SYNs (see the last paragraph of Section 3.2.1.2).

4.2.2.2. ECT on SYN Considered Invalid by Servers

A study conducted in Nov 2017 [Kuehlewind18] found that, of the 82% of the Alexa top 50k web servers that supported ECN, 84% disabled ECN if the IP/ECN field on the SYN was ECT0, CE or either. Given most web servers use Linux, this behaviour can most likely be traced to a patch contributed in May 2012 that was first distributed in v3.5 of the Linux kernel [strict-ecn]. The comment says "RFC3168 : 6.1.1 SYN packets must not have ECT/ECN bits set. If we receive a SYN packet with these bits set, it means a network is playing bad games with TOS bits. In order to avoid possible false congestion notifications, we disable TCP ECN negociation." Of course, some of the 84% might be due to similar code in other OSs.

For brevity we shall call this the "over-strict" ECN test, because it is over-conservative with what it accepts, contrary to Postel’s robustness principle. A robust protocol will not usually assume network mangling without comparing with the value originally sent, and one packet is not sufficient to make an assumption with such irreversible consequences anyway.

Ironically, networks rarely seem to alter the IP/ECN field on a SYN from zero to non-zero anyway. In a study conducted in Jan-May 2017 over millions of paths from vantage points in a few dozen mobile and fixed networks [Mandalari18], no such transition was observed. With such a small or non-existent incidence of this sort of network mangling, it would be preferable to report any residual problem paths so that they can be fixed.

Whatever, the widespread presence of this ’over-strict’ test proves that RFC 5562 was correct to expect that ECT would be considered invalid on SYNs. Nonetheless, it is not an insurmountable problem - the over-strict test in Linux was patched in Apr 2019 [relax-strict-ecn] and caching can work round it where previous versions of Linux are running. The prevalence of these "over-strict" ECN servers makes it challenging to cache them all. However, Section 4.2.3 below explains how a cache of limited size can alleviate this problem for a client's most popular sites.

For the future, [RFC8311] updates RFC 3168 to clarify that the IP/ECN field does not have to be zero on a SYN if documented in an experimental RFC such as the present ECN++ specification.

4.2.3. Caching Strategies for ECT on SYNs

Given the server handling of ECN on SYNs outlined in Section 4.2.2.2 above, an initiator might combine AccECN with three candidate caching strategies for setting ECT on a SYN:

(S1): Pessimistic ECT and cache successes: The initiator always requests AccECN, but by default without ECT on the SYN. Then it caches those servers that confirm that they support AccECN as 'ECT SYN OK'. On a subsequent connection to any server that supports AccECN, the initiator can then set ECT on the SYN. When connecting to other servers (non-ECN or classic ECN) it will not set ECT on the SYN, so it will not fail the 'over-strict' ECN test.

Longer term, as servers upgrade to AccECN, the initiator is still requesting AccECN, so it will add them to the cache and use ECT on subsequent SYNs to those servers. However, assuming it has to cap the size of the cache, the client will not have the benefit of ECT SYNs to those less frequently used AccECN servers expelled from its cache.

(S2): Optimistic ECT: The initiator always requests AccECN and by default sets ECT on the SYN. Then, if the server response shows it has no AccECN logic (so it cannot feed back a CE mark), the initiator conservatively behaves as if the SYN was CE-marked, by reducing its initial window.

A. No cache.

B. Cache failures: The optimistic ECT strategy can be improved by caching solely those servers that do not support AccECN as 'ECT SYN NOK'. This would include non-ECN servers and all Classic ECN servers whether 'over-strict' or not. On subsequent connections to these non-AccECN servers, the initiator will still request AccECN but not set ECT on the SYN. Then, the connection can still fall back to Classic ECN, if the server supports it, and the initiator can use its full initial window (if it has enough request data to need it).

Longer term, as servers upgrade to AccECN, the initiator will remove them from the cache and use ECT on subsequent SYNs to that server.

Where an access network operator mediates Internet access via a proxy that does not support AccECN, the optimistic ECT strategy will always fail. This scenario is more likely in mobile networks. Therefore, a mobile host could cache lack of AccECN support per attached access network operator. Whenever it attached to a new operator, it could check a well-known AccECN test server and, if it found no AccECN support, it would add a cache entry for the attached operator. It would only use ECT when neither network nor server were cached. It would only populate its per server cache when not attached to a non-AccECN proxy.

(S3): ECT by configuration: In a controlled environment, the administrator can make sure that servers support ECN-capable SYN packets. Examples of controlled environments are single-tenant DCs, and possibly multi-tenant DCs if it is assumed that each tenant mostly communicates with its own VMs.

For unmanaged environments like the public Internet, pragmatically the choice is between strategies (S1), (S2A) and (S2B). The normative specification for ECT on a SYN in Section 3.2.1 recommends the "optimistic ECT and cache failures" strategy (S2B) but the choice depends on the implementer’s motivation for using ECN++, and the deployment prevalence of different technologies and bug-fixes.

- The "pessimistic ECT and cache successes" strategy (S1) suffers from exposing the initial SYN to the prevailing loss level, even if the server supports ECT on SYNs, but only on the first connection to each AccECN server. If AccECN becomes widely deployed on servers, SYNs to those AccECN servers that are less frequently used by the client and therefore don’t fit in the cache will not benefit from ECN protection at all.

- The "optimistic ECT without a cache" strategy (S2A) is the simplest. It would satisfy the goal of an implementer who is solely interested in low latency using AccECN and ECN++ and is not concerned about fall-back to Classic ECN.

- The "optimistic ECT and cache failures" strategy (S2B) exploits ECT on SYNs from the very first attempt. But if the server turns out to be 'over-strict' it will disable ECN for the connection, but only for the first connection if it's one of the client's more popular servers that fits in the cache. If the server turns out not to support AccECN, the initiator has to conservatively limit its initial window, but again only for the first connection if it's one of the client's more popular servers (and anyway this rarely makes any difference when most client requests fit in a single packet).

Note that, if AccECN deployment grows, caching successes (S1) starts off small then grows, while caching failures (S2B) becomes large at first, then shrinks. At half-way, the size of the cache has to be capped with either approach, so the default behaviour for all the servers that do not fit in the cache is as important as the behaviour for the popular servers that do fit.

> MEASUREMENTS NEEDED: Measurements are needed to determine which strategy would be sufficient for any particular client, whether a particular client would need different strategies in different circumstances and how many occurrences of problems would be masked by how few cache entries.

Another strategy would be to send a not-ECT SYN a short delay (below the typical lowest RTT) after an ECT SYN and only accept the non-ECT connection if it returned first. This would reduce the performance penalty for those deploying ECT SYN support. However, this 'happy eyeballs' approach becomes complex when multiple optional features are all tried on the first SYN (or on multiple SYNs), so it is not recommended.

4.2.4. Argument 2: DoS Attacks

[RFC5562] says that ECT SYN packets could be misused by malicious clients to augment "the well-known TCP SYN attack". It goes on to say "a malicious host might be able to inject a large number of TCP SYN packets through a potentially congested ECN-enabled router, congesting it even further."

We assume this is a reference to the TCP SYN flood attack (see https://en.wikipedia.org/wiki/SYN_flood), which is an attack against a responder end point. We assume the idea of this attack is to use ECT to get more packets through an ECN-enabled router in preference to other non-ECN traffic so that they can go on to use the SYN flooding attack to inflict more damage on the responder end point. This argument could apply to flooding with any type of packet, but we assume SYNs are singled out because their source address is easier to spoof, whereas floods of other types of packets are easier to block.

Mandating Not-ECT in an RFC does not stop attackers using ECT for flooding. Nonetheless, if a standard says SYNs are not meant to be ECT it would make it legitimate for firewalls to discard them. However this would negate the considerable benefit of ECT SYNs for compliant transports and seems unnecessary because RFC 3168 already provides the means to address this concern. In section 7, RFC 3168 says "During periods where ... the potential packet marking rate would be high, our recommendation is that routers drop packets rather then set the CE codepoint..." and this advice is repeated in [RFC7567] (section 4.2.1). This makes it harder for flooding packets to gain from ECT.

[ecn-overload] showed that ECT can only slightly augment flooding attacks relative to a non-ECT attack. It was hard to overload the link without causing the queue to grow, which in turn caused the AQM to disable ECN and switch to drop, thus negating any advantage of using ECT. This was true even with the switch-over point set to 25% drop probability (i.e. the arrival rate was 133% of the link rate).

4.3. SYN-ACKs

The proposed approach in Section 3.2.2 for experimenting with ECN-capable SYN-ACKs is effectively identical to the scheme called ECN+ [ECN-PLUS]. In 2005, the ECN+ paper demonstrated that it could reduce the average Web response time by an order of magnitude. It also argued that adding ECT to SYN-ACKs did not raise any new security vulnerabilities.

4.3.1. Possibility of Unrecognized CE on the SYN-ACK

The feedback behaviour by the initiator in response to a CE-marked SYN-ACK from the responder depends on whether classic ECN feedback [RFC3168] or AccECN feedback [I-D.ietf-tcpm-accurate-ecn] has been negotiated. In either case no change is required to RFC 3168 or the AccECN specification.

Some classic ECN client implementations might ignore a CE-mark on a SYN-ACK, or even ignore a SYN-ACK packet entirely if it is set to ECT or CE. This is a possibility because an RFC 3168 implementation would not necessarily expect a SYN-ACK to be ECN-capable. This issue already came up when the IETF first decided to experiment with ECN on SYN-ACKs [RFC5562] and it was decided to go ahead without any extra precautionary measures. This was because the probability of encountering the problem was believed to be low and the harm if the problem arose was also low (see Appendix B of RFC 5562).

4.3.2. Response to Congestion on a SYN-ACK

The IETF has already specified an experiment with ECN-capable SYN-ACK packets [RFC5562]. It was inspired by the ECN+ paper, but it specified a much more conservative congestion response to a CE-marked SYN-ACK, called ECN+/TryOnce. This required the server to reduce its initial window to 1 segment (like ECN+), but then the server had to send a second SYN-ACK and wait for its ACK before it could continue with its initial window of 1 SMSS. The second SYN-ACK of this 5-way handshake had to carry no data, and had to disable ECN, but no justification was given for these last two aspects.

The present ECN++ experimental specification obsoletes RFC 5562 because it uses the ECN+ congestion response, not ECN+/TryOnce. First we argue against the rationale for ECN+/TryOnce given in sections 4.4 and 6.2 of [RFC5562]. It starts with a rather too literal interpretation of the requirement in RFC 3168 that says TCP's response to a single CE mark has to be "essentially the same as the congestion control response to a *single* dropped packet." TCP's response to a dropped initial (SYN or SYN-ACK) packet is to wait for the retransmission timer to expire (currently 1s). However, this long delay assumes the worst case between two possible causes of the loss: a) heavy overload; or b) the normal capacity-seeking behaviour of other TCP flows. When the network is still delivering CE-marked packets, it implies that there is an AQM at the bottleneck and that it is not overloaded. This is because an AQM under overload will disable ECN (as recommended in section 7 of RFC 3168 and repeated in section 4.2.1 of RFC 7567). So scenario (a) can be ruled out. Therefore, TCP's response to a CE-marked SYN-ACK can be similar to its response to the loss of _any_ packet, rather than backing off as if the special _initial_ packet of a flow has been lost.

How TCP responds to the loss of any single packet depends what it has just been doing. But there is not really a precedent for TCP's response when it experiences a CE mark having sent only one (small) packet. If TCP had been adding one segment per RTT, it would have halved its congestion window, but it hasn't established a congestion window yet. If it had been exponentially increasing it would have exited slow start, but it hasn't started exponentially increasing yet so it hasn't established a slow-start threshold.

Therefore, we have to work out a reasoned argument for what to do. If an AQM is CE-marking packets, it implies there is already a queue and it is probably already somewhere around the AQM's operating point - it is unlikely to be well below and it might be well above. So, the more data packets that the client sends in its IW, the more likely at least one will be CE marked, leading it to exit slow-start early. On the other hand, it is highly unlikely that the SYN-ACK itself pushed the AQM into congestion, so it will be safe to introduce another single segment immediately (1 RTT after the SYN-ACK). Therefore, starting to probe for capacity with a slow start from an initial window of 1 segment seems appropriate to the circumstances. This is the approach adopted in Section 3.2.2.

> EXPERIMENTATION NEEDED: Experiments will be needed to check the above reasoning and determine any better strategy for reducing IW in response to congestion on a SYN-ACK (or a SYN).

4.3.3. Fall-Back if ECT SYN-ACK Fails

An alternative to the server caching failed connection attempts would be for the server to rely on the client caching failed attempts (on the basis that the client would cache a failure whether ECT was blocked on the SYN or the SYN-ACK). This strategy cannot be used if the SYN does not request AccECN support. It works as follows: if the server receives a SYN that requests AccECN support but is set to not-ECT, it replies with a SYN-ACK also set to not-ECT. If a middlebox only blocks ECT on SYNs, not SYN-ACKs, this strategy might disable ECN on a SYN-ACK when it did not need to, but at least it saves the server from maintaining a cache.

4.4. Pure ACKs

Section 5.2 of RFC 3168 gives the following arguments for not allowing the ECT marking of pure ACKs (ACKs not piggy-backed on data):

> "To ensure the reliable delivery of the congestion indication of the CE codepoint, an ECT codepoint MUST NOT be set in a packet unless the loss of that packet in the network would be detected by the end nodes and interpreted as an indication of congestion.
>
> Transport protocols such as TCP do not necessarily detect all packet drops, such as the drop of a "pure" ACK packet; for example, TCP does not reduce the arrival rate of subsequent ACK packets in response to an earlier dropped ACK packet. Any proposal for extending ECN-Capability to such packets would have to address issues such as the case of an ACK packet that was marked with the CE codepoint but was later dropped in the network. We believe that this aspect is still the subject of research, so this document specifies that at this time, "pure" ACK packets MUST NOT indicate ECN-Capability."

Later on, in section 6.1.4 it reads:

> "For the current generation of TCP congestion control algorithms, pure acknowledgement packets (e.g., packets that do not contain any accompanying data) MUST be sent with the not-ECT codepoint. Current TCP receivers have no mechanisms for reducing traffic on the ACK-path in response to congestion notification. Mechanisms for responding to congestion on the ACK-path are areas for current and future research. (One simple possibility would be for the sender to reduce its congestion window when it receives a pure ACK packet with the CE codepoint set). For current TCP implementations, a single dropped ACK generally has only a very small effect on the TCP's sending rate."

We next address each of the arguments presented above.

The first argument is a specific instance of the reliability argument for the case of pure ACKs. This has already been addressed by countering the general reliability argument in Section 4.1.

The second argument says that ECN ought not to be enabled unless there is a mechanism to respond to it. This argument actually comprises three sub-arguments:

Mechanism feasibility: If ECN is enabled on Pure ACKs, are there, or could there be, suitable mechanisms to detect, feed back and respond to ECN-marked Pure ACKs?

Do no extra harm: There has never been a mechanism to respond to loss of non-ECN Pure ACKs. So it seems that adding ECN without a response mechanism will do no extra harm to others, while improving a connection's own performance (because loss of an ACK holds back new data). However, if the end systems have no response mechanism, ECN Pure ACKs do slightly more harm than non-ECN, because the AQM doesn't immediately clear ECT packets from the queue until it reaches overload and disables ECN.

Standards policy: Even if there were no harm to others, does it set an undesirable precedent to allow a flow to use ECN to protect its Pure ACKs from loss, when there is no mechanism to respond to ECN-marking?

The last two arguments involve value judgements, but they both depend on the concrete technical question of mechanism feasibility, which will therefore be addressed first in Section 4.4.1 below. Then Section 4.4.2 draws conclusions by addressing the value judgements in the other two questions.

4.4.1. Mechanisms to Respond to CE-Marked Pure ACKs

The question of whether the receiver of pure ACKs is required to detect and feed back any CE-marking is outside the scope of the present specification - it is a matter for the relevant feedback specification (classic ECN [RFC3168] and AccECN [I-D.ietf-tcpm-accurate-ecn]). The response to congestion feedback is also out of scope, because it would be defined in the base TCP congestion control specification [RFC5681] or its variants.

Nonetheless, in order to decide whether the present ECN++ experimental specification should require a host to set ECT on pure ACKs, we only need to know whether a response mechanism would be feasible - we do not have to standardize it. So the bullets below assess, for each type of feedback, whether the three stages of the congestion response mechanism could all work.

Detection: Can the receiver of a pure ACK detect a CE marking on it?:

* Classic feedback: RFC 3168 is silent on this point. The implementer of the receiver would not expect CE marks on pure ACKs, but the implementation might happen to check for CE marks before it looks for the data. So detection will be implementation-dependent.

* AccECN feedback: the AccECN specification requires the receiver of any TCP packets to count any CE marks on them (whether or not it sends ECN-capable control packets itself).

Feedback: TCP never ACKs a pure ACK, but the receiver of a CE-mark on a pure ACK could feed it back when it sends a subsequent data segment (if it ever does):

* Classic feedback: RFC 3168 is silent on this point, so feedback of CE-markings might be implementation specific. If the receiver (of the pure ACKs) did generate feedback, it would set the echo congestion experienced (ECE) flag in the TCP header of subsequent packets in the round, as it would to feed back CE on data packets.

* AccECN feedback: the receiver continually feeds back a count of the number of CE-marked packets that it has received and, optionally, a count of CE-marked bytes. For either metric, AccECN includes pure ACKs and indeed all types of packets.

Congestion response: In either case (classic or AccECN feedback), if the TCP sender does receive feedback about CE-markings on pure

ACKs, it will be able to reduce the congestion window (cwnd) and/ or the ACK rate.

Therefore a congestion response mechanism is clearly feasible if AccECN has been negotiated, but the position is unknown for the installed base of classic ECN feedback.

4.4.1.1. Congestion Window Response to CE-Marked Pure ACKs

This subsection explores issues that congestion control designers will need to consider when defining a cwnd response to CE-marked Pure ACKs.

A CE-mark on a Pure ACK does not mean that only Pure ACKs are causing congestion. It only means that the marked Pure ACK is part of an aggregate that is collectively causing a bottleneck queue to randomly CE-mark a fraction of the packets. A CE-mark on a Pure ACK might be due to data packets in other flows through the same bottleneck, due to data packets interspersed between Pure ACKs in the same half-connection, or just due to the rate of Pure ACKs alone. (RFC 3168 only considered the last possibility, which led to the argument that ECN-enabled Pure ACKs had to be deferred, because ACK congestion control was a research issue.)

If a host has been sending a mix of Pure ACKs and data, it doesn’t need to work out whether a particular CE mark was on a Pure ACK or not; it just needs to respond to congestion feedback as a whole by reducing its congestion window (cwnd), which limits the data it can launch into flight through the congested bottleneck. If it is purely receiving data and sending only Pure ACKs, reducing cwnd will have caused it no harm, having no effect on its ACK rate (the next subsection addresses that).

However, when a host is sending data as well as Pure ACKs, it would not be right for CE-marks on Pure ACKs and on data packets to induce the same reduction in cwnd. A possible way to address this issue would be to weight the response by the size of the marked packets (assuming the congestion control supports a weighted response, e.g. [RFC8257]). For instance, one could calculate the fraction of CE-marked bytes (headers and data) over each round trip (say) as follows:

> (CE-marked header bytes + CE-marked data bytes) / (all header bytes + all data bytes)

Header bytes can be calculated by multiplying a packet count by a nominal header size, which is possible with AccECN feedback, because it gives a count of CE-marked packets (as well as CE-marked bytes).

The above simple aggregate calculation caters for the full range of scenarios; from all Pure ACKs to just a few interspersed with data packets.

Note that any mechanism that reduces cwnd due to CE-marked Pure ACKs would need to be integrated with the congestion window validation mechanism [RFC7661], which already conservatively reduces cwnd over time because cwnd becomes stale if it is not used to fill the pipe.

4.4.1.2. ACK Rate Response to CE-Marked Pure ACKs

Reducing the congestion window will have no effect on the rate of pure ACKs. The worst case here is if the bottleneck is congested solely with pure ACKs, but it could also be problematic if a large fraction of the load was from unresponsive ACKs, leaving little or no capacity for the load from responsive data.

Since RFC 3168 was published, experimental Acknowledgement Congestion Control (AckCC) techniques have been documented in [RFC5690] (informational). So any pair of TCP end-points can choose to agree to regulate the delayed ACK ratio in response to lost or CE-marked pure ACKs. However, the protocol has a number of open issues concerning deployment (e.g. it requires support from both ends, it relies on two new TCP options, one of which is required on the SYN where option space is at a premium and, if either option is blocked by a middlebox, no fall-back behaviour is specified).

The new TCP options address two problems, namely that TCP had: i) no mechanism to allow ECT to be set on pure ACKs; and ii) no mechanism to feed back loss or CE-marking of pure ACKs. A combination of the present specification and AccECN addresses both these problems, at least for CE-marking. So it might now be possible to design an ECN-specific ACK congestion control scheme without the extra TCP options proposed in RFC 5690. However, such a mechanism is out of scope of the present document.

Setting aside the practicality of RFC 5690, the need for AckCC has not been conclusively demonstrated. It has been argued that the Internet has survived so far with no mechanism to even detect loss of pure ACKs. However, it has also been argued that ECN is not the same as loss. Packet discard can naturally thin the ACK load to whatever the bottleneck can support, whereas ECN marking does not (it queues the ACKs instead). Nonetheless, RFC 3168 (section 7) recommends that an AQM switches over from ECN marking to discard when the marking probability becomes high. Therefore discard can still be relied on to thin out ECN-enabled pure ACKs as a last resort.

4.4.2. Summary: Enabling ECN on Pure ACKs

In the case when AccECN has been negotiated, it provides a feasible congestion response mechanism, so the arguments for ECT on pure ACKs heavily outweigh those against. ECN is always more and never less reliable for delivery of congestion notification. A cwnd reduction needs to be considered by congestion control designers as a response to congestion on pure ACKs. Separately, AckCC (or an improved variant exploiting AccECN) could optionally be used to regulate the spacing between pure ACKs. However, it is not clear whether AckCC is justified. If it is not, packet discard will still act as the "congestion response of last resort" by thinning out the traffic. In contrast, not setting ECT on pure ACKs is certainly detrimental to performance, because when a pure ACK is lost it can prevent the release of new data.

In the case when Classic ECN has been negotiated, the argument for ECT on pure ACKs is less clear-cut. Some of the installed base of RFC 3168 implementations might happen to (unintentionally) provide a feedback mechanism to support a cwnd response. For those that did not, setting ECT on pure ACKs would be better for the flow’s own performance than not setting it. However, where there was no feedback mechanism, setting ECT could do slightly more harm than not setting it. AckCC could provide a complementary response mechanism, because it is designed to work with RFC 3168 ECN, but it has deployment challenges. In summary, a congestion response mechanism is unlikely to be feasible with the installed base of classic ECN.

This specification uses a safe approach. Allowing hosts to set ECT on Pure ACKs without a feasible response mechanism could result in risk. It would certainly improve the flow’s own performance, but it would slightly increase potential harm to others. Morevoer, if would set an undesirable precedent for setting ECT on packets with no mechanism to respond to any resulting congestion signals. Therefore, Section 3.2.3 allows ECT on Pure ACKs if AccECN feedback has been negotiated, but not with classic RFC 3168 ECN feedback.

4.5. Window Probes

Section 6.1.6 of RFC 3168 presents only the reliability argument for prohibiting ECT on Window probes:

> "If a window probe packet is dropped in the network, this loss is not detected by the receiver. Therefore, the TCP data sender MUST NOT set either an ECT codepoint or the CWR bit on window probe packets.

> However, because window probes use exact sequence numbers, they cannot be easily spoofed in denial-of-service attacks. Therefore, if a window probe arrives with the CE codepoint set, then the receiver SHOULD respond to the ECN indications."

The reliability argument has already been addressed in Section 4.1.

Allowing ECT on window probes could considerably improve performance because, once the receive window has reopened, if a window probe is lost the sender will stall until the next window probe reaches the receiver, which might be after the maximum retransmission timeout (at least 1 minute [RFC6928]).

On the bright side, RFC 3168 at least specifies the receiver behaviour if a CE-marked window probe arrives, so changing the behaviour ought to be less painful than for other packet types.

4.6. FINs

RFC 3168 is silent on whether a TCP sender can set ECT on a FIN. A FIN is considered as part of the sequence of data, and the rate of pure ACKs sent after a FIN could be controlled by a CE marking on the FIN. Therefore there is no reason not to set ECT on a FIN.

4.7. RSTs

RFC 3168 is silent on whether a TCP sender can set ECT on a RST. The host generating the RST message does not have an open connection after sending it (either because there was no such connection when the packet that triggered the RST message was received or because the packet that triggered the RST message also triggered the closure of the connection).

Moreover, the receiver of a CE-marked RST message can either: i) accept the RST message and close the connection; ii) emit a so-called challenge ACK in response (with suitable throttling) [RFC5961] and otherwise ignore the RST (e.g. because the sequence number is in-window but not the precise number expected next); or iii) discard the RST message (e.g. because the sequence number is out-of-window). In the first two cases there is no point in echoing any CE mark received because the sender closed its connection when it sent the RST. In the third case it makes sense to discard the CE signal as well as the RST.

Although a congestion response following a CE-marking on a RST does not appear to make sense, the following factors have been considered before deciding whether the sender ought to set ECT on a RST message:

- As explained above, a congestion response by the sender of a CE-marked RST message is not possible;
- So the only reason for the sender setting ECT on a RST would be to improve the reliability of the message's delivery;
- RST messages are used to both mount and mitigate attacks:
  * Spoofed RST messages are used by attackers to terminate ongoing connections, although the mitigations in RFC 5961 have considerably raised the bar against off-path RST attacks;
  * Legitimate RST messages allow endpoints to inform their peers to eliminate existing state that correspond to non existing connections, liberating resources e.g. in DoS attacks scenarios;
- AQMs are advised to disable ECN marking during persistent overload, so:
  * it is harder for an attacker to exploit ECN to intensify an attack;
  * it is harder for a legitimate user to exploit ECN to more reliably mitigate an attack
- Prohibiting ECT on a RST would deny the benefit of ECN to legitimate RST messages, but not to attackers who can disregard RFCs;
- If ECT were prohibited on RSTs
  * it would be easy for security middleboxes to discard all ECN-capable RSTs;
  * However, unlike a SYN flood, it is already easy for a security middlebox (or host) to distinguish a RST flood from legitimate traffic [RFC5961], and even if a some legitimate RSTs are accidentally removed as well, legitimate connections still function.

So, on balance, it has been decided that it is worth experimenting with ECT on RSTs. During experiments, if the ECN capability on RSTs is found to open a vulnerability that is hard to close, this decision can be reversed, before it is specified for the standards track.

4.8. Retransmitted Packets.

RFC 3168 says the sender "MUST NOT" set ECT on retransmitted packets. The rationale for this consumes nearly 2 pages of RFC 3168, so the reader is referred to section 6.1.5 of RFC 3168, rather than quoting it all here. There are essentially three arguments, namely: reliability; DoS attacks; and over-reaction to congestion. We address them in order below.

The reliability argument has already been addressed in Section 4.1.

Protection against DoS attacks is not afforded by prohibiting ECT on retransmitted packets. An attacker can set CE on spoofed retransmissions whether or not it is prohibited by an RFC. Protection against the DoS attack described in section 6.1.5 of RFC 3168 is solely afforded by the requirement that "the TCP data receiver SHOULD ignore the CE codepoint on out-of-window packets". Therefore in Section 3.2.7 the sender is allowed to set ECT on retransmitted packets, in order to reduce the chance of them being dropped. We also strengthen the receiver's requirement from "SHOULD ignore" to "MUST ignore". And we generalize the receiver's requirement to include failure of any validity check, not just out-of-window checks, in order to include the more stringent validity checks in RFC 5961 that have been developed since RFC 3168.

A consequence is that, for those retransmitted packets that arrive at the receiver after the original packet has been properly received (so-called spurious retransmissions), any CE marking will be ignored. There is no problem with that because the fact that the original packet has been delivered implies that the sender's original congestion response (when it deemed the packet lost and retransmitted it) was unnecessary.

Finally, the third argument is about over-reacting to congestion. The argument goes that, if a retransmitted packet is dropped, the sender will not detect it, so it will not react again to congestion (it would have reduced its congestion window already when it retransmitted the packet). Whereas, if retransmitted packets can be CE tagged instead of dropped, senders could potentially react more than once to congestion. However, we argue that it is legitimate to respond again to congestion if it still persists in subsequent round trip(s).

Therefore, in all three cases, it is not incorrect to set ECT on retransmissions.

4.9. General Fall-back for any Control Packet

Extensive experiments have found no evidence of any traversal problems with ECT on any TCP control packet [Mandalari18]. Nonetheless, Sections 3.2.1.4 and 3.2.2.3 specify fall-back measures if ECT on the first packet of each half-connection (SYN or SYN-ACK) appears to be blocking progress. Here, the question of fall-back measures for ECT on other control packets is explored. It supports the advice given in Section 3.2.8; until there's evidence that something's broken, don't fix it.

If an implementation has had to disable ECT to ensure the first packet of a flow (SYN or SYN-ACK) gets through, the question arises whether it ought to disable ECT on all subsequent control packets within the same TCP connection. Without evidence of any such problems, this seems unnecessarily cautious. Particularly given it would be hard to detect loss of most other types of TCP control packets that are not ACK'd. And particularly given that unnecessarily removing ECT from other control packets could lead to performance problems, e.g. by directing them into another queue [I-D.ietf-tsvwg-ecn-l4s-id] or over a different path, because some broken multipath equipment (erroneously) routes based on all 8 bits of the Diffserv field.

In the case where a connection starts without ECT on the SYN (perhaps because problems with previous connections had been cached), there will have been no test for ECT traversal in the client-server direction until the pure ACK that completes the handshake. It is possible that some middlebox might block ECT on this pure ACK or on later retransmissions of lost packets. Similarly, after a route change, the new path might include some middlebox that blocks ECT on some or all TCP control packets. However, without evidence of such problems, the complexity of a fix does not seem worthwhile.

> MORE MEASUREMENTS NEEDED (?): If further two-ended measurements do find evidence for these traversal problems, measurements would be needed to check for correlation of ECT traversal problems between different control packets. It might then be necessary to introduce a catch-all fall-back rule that disables ECT on certain subsequent TCP control packets based on some criteria developed from these measurements.

5. Interaction with popular variants or derivatives of TCP

The following subsections discuss any interactions between setting ECT on all packets and using the following popular variants of TCP: IW10 and TFO. It also briefly notes the possibility that the principles applied here should translate to protocols derived from

TCP. This section is informative not normative, because no interactions have been identified that require any change to specifications. The subsection on IW10 discusses potential changes to specifications but recommends that no changes are needed.

The designs of the following TCP variants have also been assessed and found not to interact adversely with ECT on TCP control packets: SYN cookies (see Appendix A of [RFC4987] and section 3.1 of [RFC5562]), TCP Fast Open (TFO [RFC7413]) and L4S [I-D.ietf-tsvwg-l4s-arch].

5.1. IW10

IW10 is an experiment to determine whether it is safe for TCP to use an initial window of 10 SMSS [RFC6928].

This subsection does not recommend any additions to the present specification in order to interwork with IW10. The specifications as they stand are safe, and there is only a corner-case with ECT on the SYN where performance could be occasionally improved, as explained below.

As specified in Section 3.2.1.1, a TCP initiator will typically only set ECT on the SYN if it requests AccECN support. If, however, the SYN-ACK tells the initiator that the responder does not support AccECN, Section 3.2.1.1 advises the initiator to conservatively reduce its initial window, preferably to 1 SMSS because, if the SYN was CE-marked, the SYN-ACK has no way to feed that back.

If the initiator implements IW10, it seems rather over-conservative to reduce IW from 10 to 1 just in case a congestion marking was missed. Nonetheless, a reduction to 1 SMSS will rarely harm performance, because:

- as long as the initiator is caching failures to negotiate AccECN, subsequent attempts to access the same server will not use ECT on the SYN anyway, so there will no longer be any need to conservatively reduce IW;

- currently, at least for web sessions, it is extremely rare for a TCP initiator (client) to have more than one data segment to send at the start of a TCP connection (see Fig 3 in [Manzoor17]) - IW10 is primarily exploited by TCP servers.

If a responder receives feedback that the SYN-ACK was CE-marked, Section 3.2.2.2 recommends that it reduces its initial window, preferably to 1 SMSS. When the responder also implements IW10, it might again seem rather over-conservative to reduce IW from 10 to 1. But in this case the rationale is somewhat different:

- Feedback that the SYN-ACK was CE-marked is an explicit indication that the queue has been building, not just uncertainty due to absence of feedback;

- Given it is now likely that a queue already exists, the more data packets that the server sends in its IW, the more likely at least one will be CE marked, leading it to exit slow-start early.

Experimentation will be needed to determine the best strategy. It should be noted that experience from recent congestion avoidance experiments where the window is reduced by less than half is not necessarily applicable to a flow start scenario. Reducing cwnd by less is one thing. Reducing an increase in cwnd by less is another.

5.2. TFO

TCP Fast Open (TFO [RFC7413]) is an experiment to remove the round trip delay of TCP's 3-way hand-shake (3WHS). A TFO initiator caches a cookie from a previous connection with a TFO-enabled server. Then, for subsequent connections to the same server, any data included on the SYN can be passed directly to the server application, which can then return up to an initial window of response data on the SYN-ACK and on data segments straight after it, without waiting for the ACK that completes the 3WHS.

The TFO experiment and the present experiment to add ECN-support for TCP control packets can be combined without altering either specification, which is justified as follows:

- The handling of ECN marking on a SYN is no different whether or not it carries data.

- In response to any CE-marking on the SYN-ACK, the responder adopts the normal response to congestion, as discussed in Section 7.2 of [RFC7413].

5.3. L4S

A Low Latency Low Loss Scalable throughput (L4S) variant of TCP such as TCP Prague [PragueLinux] is mandated to negotiate AccECN feedback, and strongly recommended to use ECN++ [I-D.ietf-tsvwg-ecn-l4s-id].

The L4S experiment and the present ECN++ experiment can be combined without altering any of the specifications. The only difference would be in the recommendation of the best SYN cache strategy.

The normative specification for ECT on a SYN in Section 3.2.1 recommends the "optimistic ECT and cache failures" strategy (S2B defined in Section 4.2.3) for the general Internet. However, if a user's Internet access bottleneck supported L4S ECN but not Classic ECN, the "optimistic ECT without a cache" strategy (S2A) would make most sense, because there would be little point trying to avoid the 'over-strict' test and negotiate Classic ECN, if L4S ECN but not Classic ECN was available on that user's access link (as is the case with Low Latency DOCSIS [DOCSIS3.1]).

Strategy (S2A) is the simplest, because it requires no cache. It would satisfy the goal of an implementer who is solely interested in ultra-low latency using AccECN and ECN++ (e.g. accessing L4S servers) and is not concerned about fall-back to Classic ECN (e.g. when accessing other servers).

5.4. Other transport protocols

Experience from experiments on adding ECN support to all TCP packets ought to be directly transferable between TCP and other transport protocols, like SCTP or QUIC.

Stream Control Transmission Protocol (SCTP [RFC4960]) is a standards track transport protocol derived from TCP. SCTP currently does not include ECN support, but Appendix A of RFC 4960 broadly describes how it would be supported and a (long-expired) draft on the addition of ECN to SCTP has been produced [I-D.stewart-tsvwg-sctpecn]. This draft avoided setting ECT on control packets and retransmissions, closely following the arguments in RFC 3168.

QUIC [I-D.ietf-quic-transport] is another standards track transport protocol offering similar services to TCP but intended to exploit some of the benefits of running over UDP. Building on the arguments in the current draft, a QUIC sender sets ECT(0) on all packets.

6. Security Considerations

Section 3.2.6 considers the question of whether ECT on RSTs will allow RST attacks to be intensified. There are several security arguments presented in RFC 3168 for preventing the ECN marking of TCP control packets and retransmitted segments. We believe all of them have been properly addressed in Section 4, particularly Section 4.2.4 and Section 4.8 on DoS attacks using spoofed ECT-marked SYNs and spoofed CE-marked retransmissions.

7. IANA Considerations

There are no IANA considerations in this memo.

8. Acknowledgments

Thanks to Mirja Kuehlewind, David Black, Padma Bhooma, Gorry Fairhurst, Michael Scharf, Yuchung Cheng and Christophe Paasch for their useful reviews.

The work of Marcelo Bagnulo has been performed in the framework of the H2020-ICT-2014-2 project 5G NORMA. His contribution reflects the consortium's view, but the consortium is not liable for any use that may be made of any of the information contained therein.

Bob Briscoe's contribution was partly funded by the Research Council of Norway through the TimeIn project, partly by CableLabs and partly by the Comcast Innovation Fund. The views expressed here are solely those of the authors.

9. References

9.1. Normative References

[I-D.ietf-tcpm-accurate-ecn]
Briscoe, B., Kuehlewind, M., and R. Scheffenegger, "More Accurate ECN Feedback in TCP", draft-ietf-tcpm-accurate-ecn-11 (work in progress), March 2020.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <https://www.rfc-editor.org/info/rfc5961>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8311] Black, D., "Relaxing Restrictions on Explicit Congestion Notification (ECN) Experimentation", RFC 8311, DOI 10.17487/RFC8311, January 2018, <https://www.rfc-editor.org/info/rfc8311>.

9.2. Informative References

[DOCSIS3.1] CableLabs, "MAC and Upper Layer Protocols Interface (MULPI) Specification, CM-SP-MULPIv3.1", Data-Over-Cable Service Interface Specifications DOCSIS(R) 3.1 Version i17 or later, January 2019, <https://specification-search.cablelabs.com/CM-SP-MULPIv3.1>.

[ecn-overload] Steen, H., "Destruction Testing: Ultra-Low Delay using Dual Queue Coupled Active Queue Management", Masters Thesis, Uni Oslo , May 2017, <https://www.duo.uio.no/bitstream/handle/10852/57424/thesis-henrste.pdf?sequence=1>.

[ecn-pam] Trammell, B., Kuehlewind, M., Boppart, D., Learmonth, I., Fairhurst, G., and R. Scheffenegger, "Enabling Internet-Wide Deployment of Explicit Congestion Notification", Int'l Conf. on Passive and Active Network Measurement (PAM'15) pp193-205, 2015, <https://link.springer.com/chapter/10.1007/978-3-319-15509-8_15>.

[ECN-PLUS] Kuzmanovic, A., "The Power of Explicit Congestion Notification", ACM SIGCOMM 35(4):61--72, 2005, <http://dl.acm.org/citation.cfm?id=1080100>.

[I-D.ietf-quic-transport] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-32 (work in progress), October 2020.

[I-D.ietf-tsvwg-ecn-l4s-id] Schepper, K. and B. Briscoe, "Identifying Modified Explicit Congestion Notification (ECN) Semantics for Ultra-Low Queuing Delay (L4S)", draft-ietf-tsvwg-ecn-l4s-id-10 (work in progress), March 2020.

[I-D.ietf-tsvwg-l4s-arch]
Briscoe, B., Schepper, K., Bagnulo, M., and G. White, "Low Latency, Low Loss, Scalable Throughput (L4S) Internet Service: Architecture", draft-ietf-tsvwg-l4s-arch-07 (work in progress), October 2020.

[I-D.stewart-tsvwg-sctpecn]
Stewart, R., Tuexen, M., and X. Dong, "ECN for Stream Control Transmission Protocol (SCTP)", draft-stewart-tsvwg-sctpecn-05 (work in progress), January 2014.

[judd-nsdi]
Judd, G., "Attaining the promise and avoiding the pitfalls of TCP in the Datacenter", USENIX Symposium on Networked Systems Design and Implementation (NSDI’15) pp.145-157, May 2015, <https://www.usenix.org/node/188966>.

[Kuehlewind18]
Kuehlewind, M., Walter, M., Learmonth, I., and B. Trammell, "Tracing Internet Path Transparency", In Proc: Network Traffic Measurement and Analysis Conference (TMA) 2018 , June 2018, <http://tma.ifip.org/2018/wp-content/uploads/sites/3/2018/06/tma2018_paper12.pdf>.

[Mandalari18]
Mandalari, A., Lutu, A., Briscoe, B., Bagnulo, M., and Oe. Alay, "Measuring ECN++: Good News for ++, Bad News for ECN over Mobile", IEEE Communications Magazine , March 2018, <https://ieeexplore.ieee.org/document/8316790>.

[Manzoor17]
Manzoor, J., Drago, I., and R. Sadre, "How HTTP/2 is changing Web traffic and how to detect it", In Proc: Network Traffic Measurement and Analysis Conference (TMA) 2017 pp.1-9, June 2017, <https://ieeexplore.ieee.org/document/8002899>.

[PragueLinux]
Briscoe, B., De Schepper, K., Albisser, O., Misund, J., Tilmans, O., Kuehlewind, M., and A. Ahmed, "Implementing the ‘TCP Prague’ Requirements for Low Latency Low Loss Scalable Throughput (L4S)", Proc. Linux Netdev 0x13 , March 2019, <https://www.netdevconf.org/0x13/session.html?talk-tcp-prague-l4s>.

[relax-strict-ecn]
Tilmans, O., "tcp: Accept ECT on SYN in the presence of RFC8311", Linux netdev patch list , April 2019, <https://lore.kernel.org/patchwork/patch/1057812/>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[RFC2140] Touch, J., "TCP Control Block Interdependence", RFC 2140, DOI 10.17487/RFC2140, April 1997, <https://www.rfc-editor.org/info/rfc2140>.

[RFC3540] Spring, N., Wetherall, D., and D. Ely, "Robust Explicit Congestion Notification (ECN) Signaling with Nonces", RFC 3540, DOI 10.17487/RFC3540, June 2003, <https://www.rfc-editor.org/info/rfc3540>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <https://www.rfc-editor.org/info/rfc4987>.

[RFC5562] Kuzmanovic, A., Mondal, A., Floyd, S., and K. Ramakrishnan, "Adding Explicit Congestion Notification (ECN) Capability to TCP's SYN/ACK Packets", RFC 5562, DOI 10.17487/RFC5562, June 2009, <https://www.rfc-editor.org/info/rfc5562>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[RFC5690] Floyd, S., Arcia, A., Ros, D., and J. Iyengar, "Adding Acknowledgement Congestion Control to TCP", RFC 5690, DOI 10.17487/RFC5690, February 2010, <https://www.rfc-editor.org/info/rfc5690>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <https://www.rfc-editor.org/info/rfc6928>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF Recommendations Regarding Active Queue Management", BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015, <https://www.rfc-editor.org/info/rfc7567>.

[RFC7661] Fairhurst, G., Sathiaseelan, A., and R. Secchi, "Updating TCP to Support Rate-Limited Traffic", RFC 7661, DOI 10.17487/RFC7661, October 2015, <https://www.rfc-editor.org/info/rfc7661>.

[RFC8257] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <https://www.rfc-editor.org/info/rfc8257>.

[strict-ecn]
Dumazet, E., "tcp: be more strict before accepting ECN negociation", Linux netdev patch list , May 2012, <https://patchwork.ozlabs.org/patch/156953/>.

Authors' Addresses

Marcelo Bagnulo
Universidad Carlos III de Madrid
Av. Universidad 30
Leganes, Madrid 28911
SPAIN

Phone: 34 91 6249500
Email: marcelo@it.uc3m.es
URI: http://www.it.uc3m.es

Bob Briscoe
Independent
UK

Email: ietf@bobbriscoe.net
URI: http://bobbriscoe.net/

Internet Engineering Task Force
Internet-Draft
Obsoletes: 793, 879, 2873, 6093, 6429,
6528, 6691 (if approved)
Updates: 5961, 1122 (if approved)
Intended status: Standards Track
Expires: April 30, 2021

W. Eddy, Ed.
MTI Systems

October 27, 2020

Transmission Control Protocol (TCP) Specification
draft-ietf-tcpm-rfc793bis-19

Abstract

This document specifies the Transmission Control Protocol (TCP). TCP is an important transport layer protocol in the Internet protocol stack, and has continuously evolved over decades of use and growth of the Internet. Over this time, a number of changes have been made to TCP as it was specified in RFC 793, though these have only been documented in a piecemeal fashion. This document collects and brings those changes together with the protocol specification from RFC 793. This document obsoletes RFC 793, as well as RFCs 879, 2873, 6093, 6429, 6528, and 6691 that updated parts of RFC 793. It updates RFC 1122, and should be considered as a replacement for the portions of that document dealing with TCP requirements. It also updates RFC 5961 by adding a small clarification in reset handling while in the SYN-RECEIVED state. The TCP header control bits from RFC 793 have also been updated based on RFC 3168.

RFC EDITOR NOTE: If approved for publication as an RFC, this should be marked additionally as "STD: 7" and replace RFC 793 in that role.

Status of This Memo

Copyright Notice

Table of Contents

1. Purpose and Scope

In 1981, RFC 793 [13] was released, documenting the Transmission Control Protocol (TCP), and replacing earlier specifications for TCP that had been published in the past.

Since then, TCP has been widely implemented, and has been used as a transport protocol for numerous applications on the Internet.

For several decades, RFC 793 plus a number of other documents have combined to serve as the core specification for TCP [45]. Over time, a number of errata have been filed against RFC 793, as well as deficiencies in security, performance, and many other aspects. The number of enhancements has grown over time across many separate documents. These were never accumulated together into a comprehensive update to the base specification.

The purpose of this document is to bring together all of the IETF Standards Track changes that have been made to the base TCP functional specification and unify them into an update of RFC 793.

Some companion documents are referenced for important algorithms that are used by TCP (e.g. for congestion control), but have not been completely included in this document. This is a conscious choice, as this base specification can be used with multiple additional algorithms that are developed and incorporated separately. This document focuses on the common basis all TCP implementations must support in order to interoperate. Since some additional TCP features have become quite complicated themselves (e.g. advanced loss recovery and congestion control), future companion documents may attempt to similarly bring these together.

In addition to the protocol specification that describes the TCP segment format, generation, and processing rules that are to be implemented in code, RFC 793 and other updates also contain informative and descriptive text for readers to understand aspects of the protocol design and operation. This document does not attempt to alter or update this informative text, and is focused only on updating the normative protocol specification. This document preserves references to the documentation containing the important explanations and rationale, where appropriate.

This document is intended to be useful both in checking existing TCP implementations for conformance purposes, as well as in writing new implementations.

2. Introduction

RFC 793 contains a discussion of the TCP design goals and provides examples of its operation, including examples of connection establishment, connection termination, packet retransmion to repair losses.

This document describes the basic functionality expected in modern TCP implementations, and replaces the protocol specification in RFC 793. It does not replicate or attempt to update the introduction and philosophy content in Sections 1 and 2 of RFC 793. Other documents are referenced to provide explanation of the theory of operation, rationale, and detailed discussion of design decisions. This document only focuses on the normative behavior of the protocol.

The "TCP Roadmap" [45] provides a more extensive guide to the RFCs that define TCP and describe various important algorithms. The TCP Roadmap contains sections on strongly encouraged enhancements that improve performance and other aspects of TCP beyond the basic operation specified in this document. As one example, implementing congestion control (e.g. [31]) is a TCP requirement, but is a complex topic on its own, and not described in detail in this document, as there are many options and possibilities that do not impact basic interoperability. Similarly, most TCP implementations today include the high-performance extensions in [43], but these are not strictly required or discussed in this document. Multipath considerations for TCP are also specified separately in [51].

A list of changes from RFC 793 is contained in Section 4.

2.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [4][11] when, and only when, they appear in all capitals, as shown here.

Each use of RFC 2119 keywords in the document is individually labeled and referenced in Appendix B that summarizes implementation requirements.

Sentences using "MUST" are labeled as "MUST-X" with X being a numeric identifier enabling the requirement to be located easily when referenced from Appendix B.

Similarly, sentences using "SHOULD" are labeled with "SHLD-X", "MAY" with "MAY-X", and "RECOMMENDED" with "REC-X".

For the purposes of this labeling, "SHOULD NOT" and "MUST NOT" are labeled the same as "SHOULD" and "MUST" instances.

2.2. Key TCP Concepts

TCP provides a reliable, in-order, byte-stream service to applications.

The application byte-stream is conveyed over the network via TCP segments, with each TCP segment sent as an Internet Protocol (IP) datagram.

TCP reliability consists of detecting packet losses (via sequence numbers) and errors (via per-segment checksums), as well as correction via retransmission.

TCP supports unicast delivery of data. Anycast applications exist that successfully use TCP without modifications, though there is some risk of instability due to changes of lower-layer forwarding behavior [42].

TCP is connection-oriented, though does not inherently include a liveness detection capability.

Data flow is supported bidirectionally over TCP connections, though applications are free to send data only unidirectionally, if they so choose.

TCP uses port numbers to identify application services and to multiplex distinct flows between hosts.

A more detailed description of TCP features compared to other transport protocols can be found in Section 3.1 of [48]. Further description of the motivations for developing TCP and its role in the Internet protocol stack can be found in Section 2 of [13] and earlier versions of the TCP specification.

3. Functional Specification

3.1. Header Format

TCP segments are sent as internet datagrams. The Internet Protocol (IP) header carries several information fields, including the source and destination host addresses [1] [12]. A TCP header follows the IP headers, supplying information specific to the TCP protocol. This division allows for the existence of host level protocols other than TCP. In early development of the Internet suite of protocols, the IP header fields had been a part of TCP.

```
       0                   1                   2                   3
       0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |          Source Port          |       Destination Port        |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                        Sequence Number                        |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                    Acknowledgment Number                      |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |  Data |       |C|E|U|A|P|R|S|F|                               |
      | Offset| Rsrvd |W|C|R|C|S|S|Y|I|            Window             |
      |       |       |R|E|G|K|H|T|N|N|                               |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |           Checksum            |         Urgent Pointer        |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                    Options                    |    Padding    |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |                             Data                              |
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Note that one tick mark represents one bit position.

Figure 1: TCP Header Format

Each of the TCP header fields is described as follows:

Source Port: 16 bits

The source port number.

Destination Port: 16 bits

The destination port number.

Sequence Number: 32 bits

The sequence number of the first data octet in this segment (except when the SYN flag is set). If SYN is set the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledgment Number: 32 bits

If the ACK control bit is set, this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established, this is always sent.

Data Offset: 4 bits

The number of 32 bit words in the TCP Header. This indicates where the data begins. The TCP header (even one including options) is an integral number of 32 bits long.

Rsrvd - Reserved: 4 bits

A set of control bits reserved for future use. Must be zero in generated segments and must be ignored in received segments, if corresponding future features are unimplemented by the sending or receiving host.

The control bits are also know as "flags". Assignment is managed by IANA from the "TCP Header Flags" registry [53].

Control Bits: 8 bits (from left to right) of currently assigned control bits:

CWR: Congestion Window Reduced (see [8])
ECE: ECN-Echo (see [8])
URG: Urgent Pointer field significant
ACK: Acknowledgment field significant
PSH: Push Function (see the Send Call description in Section 3.8.1)
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: No more data from sender

Window: 16 bits

The number of data octets beginning with the one indicated in the acknowledgment field that the sender of this segment is willing to accept.

The window size MUST be treated as an unsigned number, or else large window sizes will appear like negative windows and TCP will not work (MUST-1). It is RECOMMENDED that implementations will reserve 32-bit fields for the send and receive window sizes in the connection record and do all window computations with 32 bits (REC-1).

Checksum: 16 bits

The checksum field is the 16 bit one's complement of the one's complement sum of all 16 bit words in the header and text. The checksum computation needs to ensure the 16-bit alignment of the data being summed. If a segment contains an odd number of header and text octets, alignment can be achieved by padding the last octet with zeros on its right to form a 16 bit word for checksum purposes. The pad is not transmitted as part of the segment. While computing the checksum, the checksum field itself is replaced with zeros.

The checksum also covers a pseudo header (Figure 2) conceptually prefixed to the TCP header. The pseudo header is 96 bits for IPv4 and 320 bits for IPv6. Including the pseudo header in the checksum gives the TCP connection protection against misrouted segments. This information is carried in IP headers and is transferred across the TCP/Network interface in the arguments or results of calls by the TCP implementation on the IP layer.

```
+--------+--------+--------+--------+
|           Source Address          |
+--------+--------+--------+--------+
|         Destination Address       |
+--------+--------+--------+--------+
|  zero  |  PTCL  |    TCP Length   |
+--------+--------+--------+--------+
```

Figure 2: IPv4 Pseudo Header

Psuedo header components:

Source Address: the IPv4 source address in network byte order

Destination Address: the IPv4 destination address in network byte order

zero: bits set to zero

PTCL: the protocol number from the IP header

TCP Length: the TCP header length plus the data length in octets (this is not an explicitly transmitted quantity, but is computed), and it does not count the 12 octets of the pseudo header.

For IPv6, the pseudo header is defined in Section 8.1 of RFC 8200 [12], and contains the IPv6 Source Address and Destination Address, an Upper Layer Packet Length (a 32-bit value otherwise equivalent to TCP Length in the IPv4 pseudo header), three bytes of zero-padding, and a Next Header value (differing from the IPv6 header value in the case of extension headers present in between IPv6 and TCP).

The TCP checksum is never optional. The sender MUST generate it (MUST-2) and the receiver MUST check it (MUST-3).

Urgent Pointer: 16 bits

This field communicates the current value of the urgent pointer as a positive offset from the sequence number in this segment. The urgent pointer points to the sequence number of the octet following the urgent data. This field is only be interpreted in segments with the URG control bit set.

Options: variable

Options may occupy space at the end of the TCP header and are a multiple of 8 bits in length. All options are included in the checksum. An option may begin on any octet boundary. There are two cases for the format of an option:

Case 1: A single octet of option-kind.

Case 2: An octet of option-kind (Kind), an octet of option-length, and the actual option-data octets.

The option-length counts the two octets of option-kind and option-length as well as the option-data octets.

Note that the list of options may be shorter than the data offset field might imply. The content of the header beyond the End-of-Option option must be header padding (i.e., zero).

The list of all currently defined options is managed by IANA [52], and each option is defined in other RFCs, as indicated there. That set includes experimental options that can be extended to support multiple concurrent usages [41].

A given TCP implementation can support any currently defined options, but the following options MUST be supported (MUST-4) (kind indicated in octal):

Kind	Length	Meaning
0	-	End of option list.
1	-	No-Operation.
2	4	Maximum Segment Size.

A TCP implementation MUST be able to receive a TCP option in any segment (MUST-5).
A TCP implementation MUST (MUST-6) ignore without error any TCP option it does not implement, assuming that the option has a length

field (all TCP options except End of option list and No-Operation MUST have length fields). TCP implementations MUST be prepared to handle an illegal option length (e.g., zero); a suggested procedure is to reset the connection and log the error cause (MUST-7).

Note: There is ongoing work to extend the space available for TCP options, such as [57].

Specific Option Definitions

End of Option List

```
+--------+
|00000000|
+--------+
 Kind=0
```

This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

No-Operation

```
+--------+
|00000001|
+--------+
 Kind=1
```

This option code can be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers MUST be prepared to process options even if they do not begin on a word boundary (MUST-64).

Maximum Segment Size (MSS)

```
+--------+--------+---------+--------+
|00000010|00000100|   max seg size   |
+--------+--------+---------+--------+
 Kind=2   Length=4
```

Maximum Segment Size Option Data: 16 bits

If this option is present, then it communicates the maximum receive segment size at the TCP endpoint that sends this segment. This value is limited by the IP reassembly limit. This field may be sent in the initial connection request (i.e., in segments with the SYN control bit set) and MUST NOT be sent in other segments (MUST-65). If this option is not used, any segment size is allowed. A more complete description of this option is provided in Section 3.6.1.

Other Common Options

Additional RFCs define some other commonly used options that are recommended to implement. These are the TCP Selective Acknowledgement (SACK) option [18][21], TCP Timestamp (TS) option [43], and TCP Window Scaling (WS) option [43].

Experimental TCP Options

Experimental TCP option values are defined in [24], and [41] describes the current recommended usage for these experimental values.

Padding: variable

Padding is used to ensure that the TCP header ends and data begins on a 32 bit boundary. The padding is composed of zeros.

3.2. TCP Terminology Overview

This section includes an overview of key terms needed to understand the detailed protocol operation in the rest of the document. There is a traditional glossary of terms in Section 3.10.

3.2.1. Key Connection State Variables

Before we can discuss very much about the operation of the TCP implementation we need to introduce some detailed terminology. The maintenance of a TCP connection requires the remembering of several variables. We conceive of these variables being stored in a connection record called a Transmission Control Block or TCB. Among the variables stored in the TCB are the local and remote IP addresses and port numbers, the IP security level and compartment of the connection (see Appendix A.1), pointers to the user’s send and receive buffers, pointers to the retransmit queue and to the current segment. In addition several variables relating to the send and receive sequence numbers are stored in the TCB.

Send Sequence Variables:

SND.UNA - send unacknowledged
SND.NXT - send next
SND.WND - send window
SND.UP - send urgent pointer
SND.WL1 - segment sequence number used for last window update
SND.WL2 - segment acknowledgment number used for last window update
ISS - initial send sequence number

Receive Sequence Variables:

RCV.NXT - receive next
RCV.WND - receive window
RCV.UP - receive urgent pointer
IRS - initial receive sequence number

The following diagrams may help to relate some of these variables to the sequence space.

```
                      1         2          3          4
                 ----------|----------|----------|----------
                        SND.UNA    SND.NXT    SND.UNA
                                             +SND.WND
```

```
           1 - old sequence numbers that have been acknowledged
           2 - sequence numbers of unacknowledged data
           3 - sequence numbers allowed for new data transmission
           4 - future sequence numbers that are not yet allowed
```

Figure 3: Send Sequence Space

The send window is the portion of the sequence space labeled 3 in Figure 3.

```
                          1          2          3
                      ----------|----------|----------
                             RCV.NXT    RCV.NXT
                                       +RCV.WND
```

```
           1 - old sequence numbers that have been acknowledged
           2 - sequence numbers allowed for new reception
           3 - future sequence numbers that are not yet allowed
```

Figure 4: Receive Sequence Space

The receive window is the portion of the sequence space labeled 2 in Figure 4.

There are also some variables used frequently in the discussion that take their values from the fields of the current segment.

Current Segment Variables:

```
SEG.SEQ - segment sequence number
SEG.ACK - segment acknowledgment number
SEG.LEN - segment length
SEG.WND - segment window
SEG.UP  - segment urgent pointer
```

3.2.2. State Machine Overview

A connection progresses through a series of states during its lifetime. The states are: LISTEN, SYN-SENT, SYN-RECEIVED, ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT, and the fictional state CLOSED. CLOSED is fictional because it represents the state when there is no TCB, and therefore, no connection. Briefly the meanings of the states are:

LISTEN - represents waiting for a connection request from any remote TCP peer and port.

SYN-SENT - represents waiting for a matching connection request after having sent a connection request.

SYN-RECEIVED - represents waiting for a confirming connection request acknowledgment after having both received and sent a connection request.

ESTABLISHED - represents an open connection, data received can be delivered to the user. The normal state for the data transfer phase of the connection.

FIN-WAIT-1 - represents waiting for a connection termination request from the remote TCP peer, or an acknowledgment of the connection termination request previously sent.

FIN-WAIT-2 - represents waiting for a connection termination request from the remote TCP peer.

CLOSE-WAIT - represents waiting for a connection termination request from the local user.

CLOSING - represents waiting for a connection termination request acknowledgment from the remote TCP peer.

LAST-ACK - represents waiting for an acknowledgment of the connection termination request previously sent to the remote TCP peer (this termination request sent to the remote TCP peer already included an acknowledgment of the termination request sent from the remote TCP peer).

TIME-WAIT - represents waiting for enough time to pass to be sure the remote TCP peer received the acknowledgment of its connection termination request, and to avoid new connections being impacted by delayed segments from previous connections.

CLOSED - represents no connection state at all.

A TCP connection progresses from one state to another in response to events. The events are the user calls, OPEN, SEND, RECEIVE, CLOSE, ABORT, and STATUS; the incoming segments, particularly those containing the SYN, ACK, RST and FIN flags; and timeouts.

The state diagram in Figure 5 illustrates only state changes, together with the causing events and resulting actions, but addresses neither error conditions nor actions that are not connected with state changes. In a later section, more detail is offered with respect to the reaction of the TCP implementation to events. Some state names are abbreviated or hyphenated differently in the diagram from how they appear elsewhere in the document.

NOTA BENE: This diagram is only a summary and must not be taken as the total specification. Many details are not included.

```
                               +---------+ ---------\      active OPEN
                               |  CLOSED |            \    -----------
                               +---------+<---------\   \   create TCB
                                 |     ^              \   \  snd SYN
                    passive OPEN |     |   CLOSE        \   \
                    ------------ |     | ----------       \   \
                     create TCB  |     | delete TCB         \   \
                                 V     |                      \   \
             rcv RST (note 1)  +---------+            CLOSE    |    \
          -------------------->|  LISTEN |          ---------- |     |
         /                     +---------+          delete TCB |     |
        /           rcv SYN      |     |     SEND              |     |
       /           -----------   |     |    -------            |     V
   +--------+      snd SYN,ACK  /       \   snd SYN          +--------+
   |        |<-----------------           ------------------>|        |
   |  SYN   |                    rcv SYN                     |  SYN   |
   |  RCVD  |<-----------------------------------------------|  SENT  |
   |        |                  snd SYN,ACK                   |        |
   |        |------------------           -------------------|        |
   +--------+   rcv ACK of SYN  \       /  rcv SYN,ACK       +--------+
      |         --------------   |     |   -----------
      |                x         |     |     snd ACK
      |                          V     V
      |  CLOSE                 +---------+
      | -------                |  ESTAB  |
      | snd FIN                +---------+
      |                 CLOSE    |     |    rcv FIN
      V                -------   |     |    -------
   +---------+         snd FIN  /       \   snd ACK         +---------+
   |  FIN    |<----------------          ------------------>|  CLOSE  |
   | WAIT-1  |------------------                            |   WAIT  |
   +---------+          rcv FIN  \                          +---------+
     | rcv ACK of FIN   -------   |                          CLOSE  |
     | --------------   snd ACK   |                         ------- |
     V        x                   V                         snd FIN V
   +---------+               +---------+                    +---------+
   |FINWAIT-2|               | CLOSING |                    | LAST-ACK|
   +---------+               +---------+                    +---------+
     |              rcv ACK of FIN |                 rcv ACK of FIN |
     |  rcv FIN     -------------- |    Timeout=2MSL -------------- |
     |  -------            x       V    ------------        x       V
      \ snd ACK              +---------+delete TCB          +---------+
        -------------------->|TIME-WAIT|------------------->| CLOSED  |
                             +---------+                    +---------+
```

Figure 5: TCP Connection State Diagram

The following notes apply to Figure 5:

Note 1: The transition from SYN-RECEIVED to LISTEN on receiving a RST is conditional on having reached SYN-RECEIVED after a passive open.

Note 2: An unshown transition exists from FIN-WAIT-1 to TIME-WAIT if a FIN is received and the local FIN is also acknowledged.

Note 3: A RST can be sent from any state with a corresponding transition to TIME-WAIT (see [60] for rationale). These transitions are not not explicitly shown, otherwise the diagram would become very difficult to read. Similarly, receipt of a RST from any state results in a transition to LISTEN or CLOSED, though this is also omitted from the diagram for legibility.

3.3. Sequence Numbers

A fundamental notion in the design is that every octet of data sent over a TCP connection has a sequence number. Since every octet is sequenced, each of them can be acknowledged. The acknowledgment mechanism employed is cumulative so that an acknowledgment of sequence number X indicates that all octets up to but not including X have been received. This mechanism allows for straight-forward duplicate detection in the presence of retransmission. Numbering of octets within a segment is that the first data octet immediately following the header is the lowest numbered, and the following octets are numbered consecutively.

It is essential to remember that the actual sequence number space is finite, though very large. This space ranges from 0 to 2**32 - 1. Since the space is finite, all arithmetic dealing with sequence numbers must be performed modulo 2**32. This unsigned arithmetic preserves the relationship of sequence numbers as they cycle from 2**32 - 1 to 0 again. There are some subtleties to computer modulo arithmetic, so great care should be taken in programming the comparison of such values. The symbol "=<" means "less than or equal" (modulo 2**32).

The typical kinds of sequence number comparisons that the TCP implementation must perform include:

(a) Determining that an acknowledgment refers to some sequence number sent but not yet acknowledged.

(b) Determining that all sequence numbers occupied by a segment have been acknowledged (e.g., to remove the segment from a retransmission queue).

(c) Determining that an incoming segment contains sequence numbers that are expected (i.e., that the segment "overlaps" the receive window).

In response to sending data the TCP endpoint will receive acknowledgments. The following comparisons are needed to process the acknowledgments.

SND.UNA = oldest unacknowledged sequence number

SND.NXT = next sequence number to be sent

SEG.ACK = acknowledgment from the receiving TCP peer (next sequence number expected by the receiving TCP peer)

SEG.SEQ = first sequence number of a segment

SEG.LEN = the number of octets occupied by the data in the segment (counting SYN and FIN)

SEG.SEQ+SEG.LEN-1 = last sequence number of a segment

A new acknowledgment (called an "acceptable ack"), is one for which the inequality below holds:

SND.UNA < SEG.ACK =< SND.NXT

A segment on the retransmission queue is fully acknowledged if the sum of its sequence number and length is less or equal than the acknowledgment value in the incoming segment.

When data is received the following comparisons are needed:

RCV.NXT = next sequence number expected on an incoming segments, and is the left or lower edge of the receive window

RCV.NXT+RCV.WND-1 = last sequence number expected on an incoming segment, and is the right or upper edge of the receive window

SEG.SEQ = first sequence number occupied by the incoming segment

SEG.SEQ+SEG.LEN-1 = last sequence number occupied by the incoming segment

A segment is judged to occupy a portion of valid receive sequence space if

RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND

or

RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

The first part of this test checks to see if the beginning of the segment falls in the window, the second part of the test checks to see if the end of the segment falls in the window; if the segment passes either part of the test it contains data in the window.

Actually, it is a little more complicated than this. Due to zero windows and zero length segments, we have four cases for the acceptability of an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

Note that when the receive window is zero no segments should be acceptable except ACK segments. Thus, it is possible for a TCP implementation to maintain a zero receive window while transmitting data and receiving ACKs. A TCP receiver MUST process the RST and URG fields of all incoming segments, even when the receive window is zero (MUST-66).

We have taken advantage of the numbering scheme to protect certain control information as well. This is achieved by implicitly including some control flags in the sequence space so they can be retransmitted and acknowledged without confusion (i.e., one and only one copy of the control will be acted upon). Control information is not physically carried in the segment data space. Consequently, we must adopt rules for implicitly assigning sequence numbers to control. The SYN and FIN are the only controls requiring this protection, and these controls are used only at connection opening and closing. For sequence number purposes, the SYN is considered to occur before the first actual data octet of the segment in which it occurs, while the FIN is considered to occur after the last actual data octet in a segment in which it occurs. The segment length (SEG.LEN) includes both data and sequence space occupying controls. When a SYN is present then SEG.SEQ is the sequence number of the SYN.

Initial Sequence Number Selection

A connection is defined by a pair of sockets. Connections can be reused. New instances of a connection will be referred to as incarnations of the connection. The problem that arises from this is -- "how does the TCP implementation identify duplicate segments from previous incarnations of the connection?" This problem becomes apparent if the connection is being opened and closed in quick succession, or if the connection breaks with loss of memory and is then reestablished. To support this, the TIME-WAIT state limits the rate of connection reuse, while the initial sequence number selection described below further protects against amiguity about what incarnation of a connection an incoming packet corresponds to.

To avoid confusion we must prevent segments from one incarnation of a connection from being used while the same sequence numbers may still be present in the network from an earlier incarnation. We want to assure this, even if a TCP endpoint loses all knowledge of the sequence numbers it has been using. When new connections are created, an initial sequence number (ISN) generator is employed that selects a new 32 bit ISN. There are security issues that result if an off-path attacker is able to predict or guess ISN values.

TCP Initial Sequence Numbers are generated from a number sequence that monotonically increases until it wraps, known loosely as a "clock". This clock is a 32-bit counter that typically increments at least once every roughly 4 microseconds, although it is neither assumed to be realtime nor precise, and need not persist across reboots. The clock component is intended to insure that with a Maximum Segment Lifetime (MSL), generated ISNs will be unique, since it cycles approximately every 4.55 hours, which is much longer than the MSL.

A TCP implementation MUST use the above type of "clock" for clock-driven selection of initial sequence numbers (MUST-8), and SHOULD generate its Initial Sequence Numbers with the expression:

ISN = M + F(localip, localport, remoteip, remoteport, secretkey)

where M is the 4 microsecond timer, and F() is a pseudorandom function (PRF) of the connection's identifying parameters ("localip, localport, remoteip, remoteport") and a secret key ("secretkey") (SHLD-1). F() MUST NOT be computable from the outside (MUST-9), or an attacker could still guess at sequence numbers from the ISN used for some other connection. The PRF could be implemented as a cryptographic hash of the concatenation of the TCP connection parameters and some secret data. For discussion of the selection of a specific hash algorithm and management of the secret key data, please see Section 3 of [38].

For each connection there is a send sequence number and a receive sequence number. The initial send sequence number (ISS) is chosen by the data sending TCP peer, and the initial receive sequence number (IRS) is learned during the connection establishing procedure.

For a connection to be established or initialized, the two TCP peers must synchronize on each other’s initial sequence numbers. This is done in an exchange of connection establishing segments carrying a control bit called "SYN" (for synchronize) and the initial sequence numbers. As a shorthand, segments carrying the SYN bit are also called "SYNs". Hence, the solution requires a suitable mechanism for picking an initial sequence number and a slightly involved handshake to exchange the ISNs.

The synchronization requires each side to send its own initial sequence number and to receive a confirmation of it in acknowledgment from the remote TCP peer. Each side must also receive the remote peer’s initial sequence number and send a confirming acknowledgment.

```
    1) A --> B  SYN my sequence number is X
    2) A <-- B  ACK your sequence number is X
    3) A <-- B  SYN my sequence number is Y
    4) A --> B  ACK your sequence number is Y
```

Because steps 2 and 3 can be combined in a single message this is called the three-way (or three message) handshake (3WHS).

A 3WHS is necessary because sequence numbers are not tied to a global clock in the network, and TCP implementations may have different mechanisms for picking the ISNs. The receiver of the first SYN has no way of knowing whether the segment was an old delayed one or not, unless it remembers the last sequence number used on the connection (which is not always possible), and so it must ask the sender to verify this SYN. The three way handshake and the advantages of a clock-driven scheme are discussed in [59].

Knowing When to Keep Quiet

A theoretical problem exists where data could be corrupted due to confusion between old segments in the network and new ones after a host reboots, if the same port numbers and sequence space are reused. The "Quiet Time" concept discussed below addresses this and the discussion of it is included for situations where it might be relevant, although it is not felt to be necessary in most current implementations. The problem was more relevant earlier in the history of TCP. In practical use on the Internet today, the error-prone conditions are sufficiently unlikely that it is felt safe to ignore. Reasons why it is now negligible include: (a) ISS and ephemeral port randomization have reduced likelihood of reuse of port numbers and sequence numbers after reboots, (b) the effective MSL of the Internet has declined as links have become faster, and (c) reboots often taking longer than an MSL anyways.

To be sure that a TCP implementation does not create a segment carrying a sequence number that may be duplicated by an old segment remaining in the network, the TCP endpoint must keep quiet for an MSL before assigning any sequence numbers upon starting up or recovering from a situation where memory of sequence numbers in use was lost. For this specification the MSL is taken to be 2 minutes. This is an engineering choice, and may be changed if experience indicates it is desirable to do so. Note that if a TCP endpoint is reinitialized in some sense, yet retains its memory of sequence numbers in use, then it need not wait at all; it must only be sure to use sequence numbers larger than those recently used.

The TCP Quiet Time Concept

Hosts that for any reason lose knowledge of the last sequence numbers transmitted on each active (i.e., not closed) connection shall delay emitting any TCP segments for at least the agreed MSL in the internet system that the host is a part of. In the paragraphs below, an explanation for this specification is given. TCP implementors may violate the "quiet time" restriction, but only at the risk of causing some old data to be accepted as new or new data rejected as old duplicated by some receivers in the internet system.

TCP endpoints consume sequence number space each time a segment is formed and entered into the network output queue at a source host. The duplicate detection and sequencing algorithm in the TCP protocol relies on the unique binding of segment data to sequence space to the extent that sequence numbers will not cycle through all 2**32 values before the segment data bound to those sequence numbers has been delivered and acknowledged by the receiver and all duplicate copies of the segments have "drained" from the internet. Without such an assumption, two distinct TCP segments could conceivably be assigned the same or overlapping sequence numbers, causing confusion at the receiver as to which data is new and which is old. Remember that each segment is bound to as many consecutive sequence numbers as there are octets of data and SYN or FIN flags in the segment.

Under normal conditions, TCP implementations keep track of the next sequence number to emit and the oldest awaiting acknowledgment so as to avoid mistakenly using a sequence number over before its first use has been acknowledged. This alone does not guarantee that old duplicate data is drained from the net, so the sequence space has been made very large to reduce the probability that a wandering duplicate will cause trouble upon arrival. At 2 megabits/sec. it takes 4.5 hours to use up 2**32 octets of sequence space. Since the maximum segment lifetime in the net is not likely to exceed a few tens of seconds, this is deemed ample protection for foreseeable nets, even if data rates escalate to 10's of megabits/sec. At 100 megabits/sec, the cycle time is 5.4 minutes, which may be a little short, but still within reason.

The basic duplicate detection and sequencing algorithm in TCP can be defeated, however, if a source TCP endpoint does not have any memory of the sequence numbers it last used on a given connection. For example, if the TCP implementation were to start all connections with sequence number 0, then upon the host rebooting, a TCP peer might re-form an earlier connection (possibly after half-open connection resolution) and emit packets with sequence numbers identical to or overlapping with packets still in the network, which were emitted on an earlier incarnation of the same connection. In the absence of knowledge about the sequence numbers used on a particular connection, the TCP specification recommends that the source delay for MSL seconds before emitting segments on the connection, to allow time for segments from the earlier connection incarnation to drain from the system.

Even hosts that can remember the time of day and used it to select initial sequence number values are not immune from this problem (i.e., even if time of day is used to select an initial sequence number for each new connection incarnation).

Suppose, for example, that a connection is opened starting with sequence number S. Suppose that this connection is not used much and that eventually the initial sequence number function (ISN(t)) takes on a value equal to the sequence number, say S1, of the last segment sent by this TCP endpoint on a particular connection. Now suppose, at this instant, the host reboots and establishes a new incarnation of the connection. The initial sequence number chosen is S1 = ISN(t) -- last used sequence number on old incarnation of connection! If the recovery occurs quickly enough, any old duplicates in the net bearing sequence numbers in the neighborhood of S1 may arrive and be treated as new packets by the receiver of the new incarnation of the connection.

The problem is that the recovering host may not know for how long it was down between rebooting nor does it know whether there are still old duplicates in the system from earlier connection incarnations.

One way to deal with this problem is to deliberately delay emitting segments for one MSL after recovery from a reboot - this is the "quiet time" specification. Hosts that prefer to avoid waiting are willing to risk possible confusion of old and new packets at a given destination may choose not to wait for the "quiet time". Implementors may provide TCP users with the ability to select on a connection by connection basis whether to wait after a reboot, or may informally implement the "quiet time" for all connections. Obviously, even where a user selects to "wait," this is not necessary after the host has been "up" for at least MSL seconds.

To summarize: every segment emitted occupies one or more sequence numbers in the sequence space, the numbers occupied by a segment are "busy" or "in use" until MSL seconds have passed, upon rebooting a block of space-time is occupied by the octets and SYN or FIN flags of the last emitted segment, if a new connection is started too soon and uses any of the sequence numbers in the space-time footprint of the last segment of the previous connection incarnation, there is a potential sequence number overlap area that could cause confusion at the receiver.

3.4. Establishing a connection

The "three-way handshake" is the procedure used to establish a connection. This procedure normally is initiated by one TCP peer and responded to by another TCP peer. The procedure also works if two TCP peers simultaneously initiate the procedure. When simultaneous open occurs, each TCP peer receives a "SYN" segment that carries no acknowledgment after it has sent a "SYN". Of course, the arrival of an old duplicate "SYN" segment can potentially make it appear, to the recipient, that a simultaneous connection initiation is in progress. Proper use of "reset" segments can disambiguate these cases.

Several examples of connection initiation follow. Although these examples do not show connection synchronization using data-carrying segments, this is perfectly legitimate, so long as the receiving TCP endpoint doesn’t deliver the data to the user until it is clear the data is valid (e.g., the data is buffered at the receiver until the connection reaches the ESTABLISHED state, given that the three-way handshake reduces the possibility of false connections). It is the implementation of a trade-off between memory and messages to provide information for this checking.

The simplest 3WHS is shown in Figure 6. The figures should be interpreted in the following way. Each line is numbered for reference purposes. Right arrows (-->) indicate departure of a TCP segment from TCP peer A to TCP peer B, or arrival of a segment at B from A. Left arrows (<--), indicate the reverse. Ellipsis (...)

indicates a segment that is still in the network (delayed). Comments appear in parentheses. TCP connection states represent the state AFTER the departure or arrival of the segment (whose contents are shown in the center of each line). Segment contents are shown in abbreviated form, with sequence number, control flags, and ACK field. Other fields such as window, addresses, lengths, and text have been left out in the interest of clarity.

```
    TCP Peer A                                           TCP Peer B

1.  CLOSED                                               LISTEN

2.  SYN-SENT    --> <SEQ=100><CTL=SYN>               --> SYN-RECEIVED

3.  ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>  <-- SYN-RECEIVED

4.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>       --> ESTABLISHED

5.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA> --> ESTABLISHED
```

Figure 6: Basic 3-Way Handshake for Connection Synchronization

In line 2 of Figure 6, TCP Peer A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100. In line 3, TCP Peer B sends a SYN and acknowledges the SYN it received from TCP Peer A. Note that the acknowledgment field indicates TCP Peer B is now expecting to hear sequence 101, acknowledging the SYN that occupied sequence 100.

At line 4, TCP Peer A responds with an empty segment containing an ACK for TCP Peer B’s SYN; and in line 5, TCP Peer A sends some data. Note that the sequence number of the segment in line 5 is the same as in line 4 because the ACK does not occupy sequence number space (if it did, we would wind up ACKing ACKs!).

Simultaneous initiation is only slightly more complex, as is shown in Figure 7. Each TCP peer’s connection state cycles from CLOSED to SYN-SENT to SYN-RECEIVED to ESTABLISHED.

```
        TCP Peer A                                           TCP Peer B

    1.  CLOSED                                               CLOSED

    2.  SYN-SENT     --> <SEQ=100><CTL=SYN>                  ...

    3.  SYN-RECEIVED <-- <SEQ=300><CTL=SYN>                  <-- SYN-SENT

    4.               ... <SEQ=100><CTL=SYN>                  --> SYN-RECEIVED

    5.  SYN-RECEIVED --> <SEQ=100><ACK=301><CTL=SYN,ACK> ...

    6.  ESTABLISHED  <-- <SEQ=300><ACK=101><CTL=SYN,ACK> <-- SYN-RECEIVED

    7.               ... <SEQ=100><ACK=301><CTL=SYN,ACK> --> ESTABLISHED
```

Figure 7: Simultaneous Connection Synchronization

A TCP implementation MUST support simultaneous open attempts (MUST-10).

Note that a TCP implementation MUST keep track of whether a connection has reached SYN-RECEIVED state as the result of a passive OPEN or an active OPEN (MUST-11).

The principal reason for the three-way handshake is to prevent old duplicate connection initiations from causing confusion. To deal with this, a special control message, reset, is specified. If the receiving TCP peer is in a non-synchronized state (i.e., SYN-SENT, SYN-RECEIVED), it returns to LISTEN on receiving an acceptable reset. If the TCP peer is in one of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), it aborts the connection and informs its user. We discuss this latter case under "half-open" connections below.

```
       TCP Peer A                                           TCP Peer B

   1.  CLOSED                                               LISTEN

   2.  SYN-SENT    --> <SEQ=100><CTL=SYN>               ...

   3.  (duplicate) ... <SEQ=90><CTL=SYN>               --> SYN-RECEIVED

   4.  SYN-SENT    <-- <SEQ=300><ACK=91><CTL=SYN,ACK>  <-- SYN-RECEIVED

   5.  SYN-SENT    --> <SEQ=91><CTL=RST>               --> LISTEN

   6.              ... <SEQ=100><CTL=SYN>               --> SYN-RECEIVED

   7.  ESTABLISHED <-- <SEQ=400><ACK=101><CTL=SYN,ACK>  <-- SYN-RECEIVED

   8.  ESTABLISHED --> <SEQ=101><ACK=401><CTL=ACK>      --> ESTABLISHED
```

Figure 8: Recovery from Old Duplicate SYN

As a simple example of recovery from old duplicates, consider Figure 8. At line 3, an old duplicate SYN arrives at TCP Peer B. TCP Peer B cannot tell that this is an old duplicate, so it responds normally (line 4). TCP Peer A detects that the ACK field is incorrect and returns a RST (reset) with its SEQ field selected to make the segment believable. TCP Peer B, on receiving the RST, returns to the LISTEN state. When the original SYN finally arrives at line 6, the synchronization proceeds normally. If the SYN at line 6 had arrived before the RST, a more complex exchange might have occurred with RST's sent in both directions.

Half-Open Connections and Other Anomalies

An established connection is said to be "half-open" if one of the TCP peers has closed or aborted the connection at its end without the knowledge of the other, or if the two ends of the connection have become desynchronized owing to a failure or reboot that resulted in loss of memory. Such connections will automatically become reset if an attempt is made to send data in either direction. However, half-open connections are expected to be unusual.

If at site A the connection no longer exists, then an attempt by the user at site B to send any data on it will result in the site B TCP endpoint receiving a reset control message. Such a message indicates to the site B TCP endpoint that something is wrong, and it is expected to abort the connection.

Assume that two user processes A and B are communicating with one another when a failure or reboot occurs causing loss of memory to A’s TCP implementation. Depending on the operating system supporting A’s TCP implementation, it is likely that some error recovery mechanism exists. When the TCP endpoint is up again, A is likely to start again from the beginning or from a recovery point. As a result, A will probably try to OPEN the connection again or try to SEND on the connection it believes open. In the latter case, it receives the error message "connection not open" from the local (A’s) TCP implementation. In an attempt to establish the connection, A’s TCP implementation will send a segment containing SYN. This scenario leads to the example shown in Figure 9. After TCP Peer A reboots, the user attempts to re-open the connection. TCP Peer B, in the meantime, thinks the connection is open.

```
      TCP Peer A                                           TCP Peer B

  1.  (REBOOT)                              (send 300,receive 100)

  2.  CLOSED                                               ESTABLISHED

  3.  SYN-SENT --> <SEQ=400><CTL=SYN>              --> (??)

  4.  (!!)     <-- <SEQ=300><ACK=100><CTL=ACK>     <-- ESTABLISHED

  5.  SYN-SENT --> <SEQ=100><CTL=RST>              --> (Abort!!)

  6.  SYN-SENT                                         CLOSED

  7.  SYN-SENT --> <SEQ=400><CTL=SYN>              -->
```

Figure 9: Half-Open Connection Discovery

When the SYN arrives at line 3, TCP Peer B, being in a synchronized state, and the incoming segment outside the window, responds with an acknowledgment indicating what sequence it next expects to hear (ACK 100). TCP Peer A sees that this segment does not acknowledge anything it sent and, being unsynchronized, sends a reset (RST) because it has detected a half-open connection. TCP Peer B aborts at line 5. TCP Peer A will continue to try to establish the connection; the problem is now reduced to the basic 3-way handshake of Figure 6.

An interesting alternative case occurs when TCP Peer A reboots and TCP Peer B tries to send data on what it thinks is a synchronized connection. This is illustrated in Figure 10. In this case, the data arriving at TCP Peer A from TCP Peer B (line 2) is unacceptable because no such connection exists, so TCP Peer A sends a RST. The

RST is acceptable so TCP Peer B processes it and aborts the connection.

```
       TCP Peer A                                            TCP Peer B

   1.  (REBOOT)                                   (send 300,receive 100)

   2.  (??)    <-- <SEQ=300><ACK=100><DATA=10><CTL=ACK> <-- ESTABLISHED

   3.          --> <SEQ=100><CTL=RST>                   --> (ABORT!!)
```

Figure 10: Active Side Causes Half-Open Connection Discovery

In Figure 11, two TCP Peers A and B with passive connections waiting for SYN are depicted. An old duplicate arriving at TCP Peer B (line 2) stirs B into action. A SYN-ACK is returned (line 3) and causes TCP A to generate a RST (the ACK in line 3 is not acceptable). TCP Peer B accepts the reset and returns to its passive LISTEN state.

```
       TCP Peer A                                       TCP Peer B

   1.  LISTEN                                           LISTEN

   2.       ... <SEQ=Z><CTL=SYN>                -->     SYN-RECEIVED

   3.  (??) <-- <SEQ=X><ACK=Z+1><CTL=SYN,ACK>   <--     SYN-RECEIVED

   4.       --> <SEQ=Z+1><CTL=RST>              -->     (return to LISTEN!)

   5.  LISTEN                                           LISTEN
```

Figure 11: Old Duplicate SYN Initiates a Reset on two Passive Sockets

A variety of other cases are possible, all of which are accounted for by the following rules for RST generation and processing.

Reset Generation

A TCP user or application can issue a reset on a connection at any time, though reset events are also generated by the protocol itself when various error conditions occur, as described below. The side of a connection issuing a reset should enter the TIME-WAIT state, as this generally helps to reduce the load on busy servers for reasons described in [60].

As a general rule, reset (RST) is sent whenever a segment arrives that apparently is not intended for the current connection. A reset must not be sent if it is not clear that this is the case.

There are three groups of states:

1. If the connection does not exist (CLOSED) then a reset is sent in response to any incoming segment except another reset. A SYN segment that does not match an existing connection is rejected by this means.

   If the incoming segment has the ACK bit set, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the CLOSED state.

2. If the connection is in any non-synchronized state (LISTEN, SYN-SENT, SYN-RECEIVED), and the incoming segment acknowledges something not yet sent (the segment carries an unacceptable ACK), or if an incoming segment has a security level or compartment that does not exactly match the level and compartment requested for the connection, a reset is sent.

   If the incoming segment has an ACK field, the reset takes its sequence number from the ACK field of the segment, otherwise the reset has sequence number zero and the ACK field is set to the sum of the sequence number and segment length of the incoming segment. The connection remains in the same state.

3. If the connection is in a synchronized state (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, TIME-WAIT), any unacceptable segment (out of window sequence number or unacceptable acknowledgment number) must be responded to with an empty acknowledgment segment (without any user data) containing the current send-sequence number and an acknowledgment indicating the next sequence number expected to be received, and the connection remains in the same state.

   If an incoming segment has a security level, or compartment that does not exactly match the level and compartment requested for the connection, a reset is sent and the connection goes to the CLOSED state. The reset takes its sequence number from the ACK field of the incoming segment.

Reset Processing

In all states except SYN-SENT, all reset (RST) segments are validated by checking their SEQ-fields. A reset is valid if its sequence number is in the window. In the SYN-SENT state (a RST received in response to an initial SYN), the RST is acceptable if the ACK field acknowledges the SYN.

The receiver of a RST first validates it, then changes state. If the receiver was in the LISTEN state, it ignores it. If the receiver was in SYN-RECEIVED state and had previously been in the LISTEN state, then the receiver returns to the LISTEN state, otherwise the receiver aborts the connection and goes to the CLOSED state. If the receiver was in any other state, it aborts the connection and advises the user and goes to the CLOSED state.

TCP implementations SHOULD allow a received RST segment to include data (SHLD-2).

3.5. Closing a Connection

CLOSE is an operation meaning "I have no more data to send." The notion of closing a full-duplex connection is subject to ambiguous interpretation, of course, since it may not be obvious how to treat the receiving side of the connection. We have chosen to treat CLOSE in a simplex fashion. The user who CLOSEs may continue to RECEIVE until the TCP receiver is told that the remote peer has CLOSED also. Thus, a program could initiate several SENDs followed by a CLOSE, and then continue to RECEIVE until signaled that a RECEIVE failed because the remote peer has CLOSED. The TCP implementation will signal a user, even if no RECEIVEs are outstanding, that the remote peer has closed, so the user can terminate his side gracefully. A TCP implementation will reliably deliver all buffers SENT before the connection was CLOSED so a user who expects no data in return need only wait to hear the connection was CLOSED successfully to know that all their data was received at the destination TCP endpoint. Users must keep reading connections they close for sending until the TCP implementation indicates there is no more data.

There are essentially three cases:

1) The user initiates by telling the TCP implementation to CLOSE the connection (TCP Peer A in Figure 12).

2) The remote TCP endpoint initiates by sending a FIN control signal (TCP Peer B in Figure 12).

3) Both users CLOSE simultaneously (Figure 13).

Case 1: Local user initiates the close

In this case, a FIN segment can be constructed and placed on the outgoing segment queue. No further SENDs from the user will be accepted by the TCP implementation, and it enters the FIN-WAIT-1 state. RECEIVEs are allowed in this state. All segments preceding and including FIN will be retransmitted until acknowledged. When the other TCP peer has both acknowledged the FIN and sent a FIN of its own, the first TCP peer can ACK this FIN. Note that a TCP endpoint receiving a FIN will ACK but not send its own FIN until its user has CLOSED the connection also.

Case 2: TCP endpoint receives a FIN from the network

If an unsolicited FIN arrives from the network, the receiving TCP endpoint can ACK it and tell the user that the connection is closing. The user will respond with a CLOSE, upon which the TCP endpoint can send a FIN to the other TCP peer after sending any remaining data. The TCP endpoint then waits until its own FIN is acknowledged whereupon it deletes the connection. If an ACK is not forthcoming, after the user timeout the connection is aborted and the user is told.

Case 3: Both users close simultaneously

A simultaneous CLOSE by users at both ends of a connection causes FIN segments to be exchanged (Figure 13). When all segments preceding the FINs have been processed and acknowledged, each TCP peer can ACK the FIN it has received. Both will, upon receiving these ACKs, delete the connection.

```
    TCP Peer A                                           TCP Peer B

1.  ESTABLISHED                                          ESTABLISHED

2.  (Close)
    FIN-WAIT-1  --> <SEQ=100><ACK=300><CTL=FIN,ACK>  --> CLOSE-WAIT

3.  FIN-WAIT-2  <-- <SEQ=300><ACK=101><CTL=ACK>      <-- CLOSE-WAIT

4.                                                       (Close)
    TIME-WAIT   <-- <SEQ=300><ACK=101><CTL=FIN,ACK>  <-- LAST-ACK

5.  TIME-WAIT   --> <SEQ=101><ACK=301><CTL=ACK>      --> CLOSED

6.  (2 MSL)
    CLOSED
```

Figure 12: Normal Close Sequence

```
       TCP Peer A                                           TCP Peer B

   1.  ESTABLISHED                                          ESTABLISHED

   2.  (Close)                                              (Close)
       FIN-WAIT-1  --> <SEQ=100><ACK=300><CTL=FIN,ACK>  ... FIN-WAIT-1
                   <-- <SEQ=300><ACK=100><CTL=FIN,ACK>  <--
                   ... <SEQ=100><ACK=300><CTL=FIN,ACK>  -->

   3.  CLOSING     --> <SEQ=101><ACK=301><CTL=ACK>      ... CLOSING
                   <-- <SEQ=301><ACK=101><CTL=ACK>      <--
                   ... <SEQ=101><ACK=301><CTL=ACK>      -->

   4.  TIME-WAIT                                            TIME-WAIT
       (2 MSL)                                              (2 MSL)
       CLOSED                                               CLOSED
```

Figure 13: Simultaneous Close Sequence

A TCP connection may terminate in two ways: (1) the normal TCP close sequence using a FIN handshake (Figure 12), and (2) an "abort" in which one or more RST segments are sent and the connection state is immediately discarded. If the local TCP connection is closed by the remote side due to a FIN or RST received from the remote side, then the local application MUST be informed whether it closed normally or was aborted (MUST-12).

3.5.1. Half-Closed Connections

The normal TCP close sequence delivers buffered data reliably in both directions. Since the two directions of a TCP connection are closed independently, it is possible for a connection to be "half closed," i.e., closed in only one direction, and a host is permitted to continue sending data in the open direction on a half-closed connection.

A host MAY implement a "half-duplex" TCP close sequence, so that an application that has called CLOSE cannot continue to read data from the connection (MAY-1). If such a host issues a CLOSE call while received data is still pending in the TCP connection, or if new data is received after CLOSE is called, its TCP implementation SHOULD send a RST to show that data was lost (SHLD-3). See [19] section 2.17 for discussion.

When a connection is closed actively, it MUST linger in the TIME-WAIT state for a time 2xMSL (Maximum Segment Lifetime) (MUST-13). However, it MAY accept a new SYN from the remote TCP endpoint to reopen the connection directly from TIME-WAIT state (MAY-2), if it:

(1) assigns its initial sequence number for the new connection to be larger than the largest sequence number it used on the previous connection incarnation, and

(2) returns to TIME-WAIT state if the SYN turns out to be an old duplicate.

When the TCP Timestamp options are available, an improved algorithm is described in [36] in order to support higher connection establishment rates. This algorithm for reducing TIME-WAIT is a Best Current Practice that SHOULD be implemented, since timestamp options are commonly used, and using them to reduce TIME-WAIT provides benefits for busy Internet servers (SHLD-4).

3.6. Segmentation

The term "segmentation" refers to the activity TCP performs when ingesting a stream of bytes from a sending application and packetizing that stream of bytes into TCP segments. Individual TCP segments often do not correspond one-for-one to individual send (or socket write) calls from the application. Applications may perform writes at the granularity of messages in the upper layer protocol, but TCP guarantees no boundary coherence between the TCP segments sent and received versus user application data read or write buffer boundaries. In some specific protocols, such as Remote Direct Memory Access (RDMA) using Direct Data Placement (DDP) and Marker PDU Aligned Framing (MPA) [28], there are performance optimizations possible when the relation between TCP segments and application data units can be controlled, and MPA includes a specific mechanism for detecting and verifying this relationship between TCP segments and application message data structures, but this is specific to applications like RDMA. In general, multiple goals influence the sizing of TCP segments created by a TCP implementation.

Goals driving the sending of larger segments include:

- Reducing the number of packets in flight within the network.

- Increasing processing efficiency and potential performance by enabling a smaller number of interrupts and inter-layer interactions.

- Limiting the overhead of TCP headers.

Note that the performance benefits of sending larger segments may decrease as the size increases, and there may be boundaries where advantages are reversed. For instance, on some implementation architectures, 1025 bytes within a segment could lead to worse performance than 1024 bytes, due purely to data alignment on copy operations.

Goals driving the sending of smaller segments include:

- Avoiding sending a TCP segment that would result in an IP datagram larger than the smallest MTU along an IP network path, because this results in either packet loss or packet fragmentation. Making matters worse, some firewalls or middleboxes may drop fragmented packets or ICMP messages related to fragmentation.

- Preventing delays to the application data stream, especially when TCP is waiting on the application to generate more data, or when the application is waiting on an event or input from its peer in order to generate more data.

- Enabling "fate sharing" between TCP segments and lower-layer data units (e.g. below IP, for links with cell or frame sizes smaller than the IP MTU).

Towards meeting these competing sets of goals, TCP includes several mechanisms, including the Maximum Segment Size option, Path MTU Discovery, the Nagle algorithm, and support for IPv6 Jumbograms, as discussed in the following subsections.

3.6.1. Maximum Segment Size Option

TCP endpoints MUST implement both sending and receiving the MSS option (MUST-14).

TCP implementations SHOULD send an MSS option in every SYN segment when its receive MSS differs from the default 536 for IPv4 or 1220 for IPv6 (SHLD-5), and MAY send it always (MAY-3).

If an MSS option is not received at connection setup, TCP implementations MUST assume a default send MSS of 536 (576-40) for IPv4 or 1220 (1280 - 60) for IPv6 (MUST-15).

The maximum size of a segment that TCP endpoint really sends, the "effective send MSS," MUST be the smaller (MUST-16) of the send MSS (that reflects the available reassembly buffer size at the remote host, the EMTU_R [15]) and the largest transmission size permitted by the IP layer (EMTU_S [15]):

```
    Eff.snd.MSS =

        min(SendMSS+20, MMS_S) - TCPhdrsize - IPoptionsize
```

where:

- o SendMSS is the MSS value received from the remote host, or the default 536 for IPv4 or 1220 for IPv6, if no MSS option is received.

- o MMS_S is the maximum size for a transport-layer message that TCP may send.

- o TCPhdrsize is the size of the fixed TCP header and any options. This is 20 in the (rare) case that no options are present, but may be larger if TCP options are to be sent. Note that some options may not be included on all segments, but that for each segment sent, the sender should adjust the data length accordingly, within the Eff.snd.MSS.

- o IPoptionsize is the size of any IP options associated with a TCP connection. Note that some options may not be included on all packets, but that for each segment sent, the sender should adjust the data length accordingly, within the Eff.snd.MSS.

The MSS value to be sent in an MSS option should be equal to the effective MTU minus the fixed IP and TCP headers. By ignoring both IP and TCP options when calculating the value for the MSS option, if there are any IP or TCP options to be sent in a packet, then the sender must decrease the size of the TCP data accordingly. RFC 6691 [39] discusses this in greater detail.

The MSS value to be sent in an MSS option must be less than or equal to:

```
MMS_R - 20
```

where MMS_R is the maximum size for a transport-layer message that can be received (and reassembled at the IP layer) (MUST-67). TCP obtains MMS_R and MMS_S from the IP layer; see the generic call GET_MAXSIZES in Section 3.4 of RFC 1122. These are defined in terms of their IP MTU equivalents, EMTU_R and EMTU_S [15].

When TCP is used in a situation where either the IP or TCP headers are not fixed, the sender must reduce the amount of TCP data in any given packet by the number of octets used by the IP and TCP options. This has been a point of confusion historically, as explained in RFC 6691, Section 3.1.

3.6.2. Path MTU Discovery

A TCP implementation may be aware of the MTU on directly connected links, but will rarely have insight about MTUs across an entire network path. For IPv4, RFC 1122 recommends an IP-layer default effective MTU of less than or equal to 576 for destinations not directly connected. For IPv6, this would be 1280. In all cases, however, implementation of Path MTU Discovery (PMTUD) and Packetization Layer Path MTU Discovery (PLPMTUD) is strongly recommended in order for TCP to improve segmentation decisions. Both PMTUD and PLPMTUD help TCP choose segment sizes that avoid both on-path (for IPv4) and source fragmentation (IPv4 and IPv6).

PMTUD for IPv4 [2] or IPv6 [3] is implemented in conjunction between TCP, IP, and ICMP protocols. It relies both on avoiding source fragmentation and setting the IPv4 DF (don’t fragment) flag, the latter to inhibit on-path fragmentation. It relies on ICMP errors from routers along the path, whenever a segment is too large to traverse a link. Several adjustments to a TCP implementation with PMTUD are described in RFC 2923 in order to deal with problems experienced in practice [7]. PLPMTUD [25] is a Standards Track improvement to PMTUD that relaxes the requirement for ICMP support across a path, and improves performance in cases where ICMP is not consistently conveyed, but still tries to avoid source fragmentation. The mechanisms in all four of these RFCs are recommended to be included in TCP implementations.

The TCP MSS option specifies an upper bound for the size of packets that can be received. Hence, setting the value in the MSS option too small can impact the ability for PMTUD or PLPMTUD to find a larger path MTU. RFC 1191 discusses this implication of many older TCP implementations setting MSS to 536 for non-local destinations, rather than deriving it from the MTUs of connected interfaces as recommended.

3.6.3. Interfaces with Variable MTU Values

The effective MTU can sometimes vary, as when used with variable compression, e.g., RObust Header Compression (ROHC) [32]. It is tempting for a TCP implementation to advertise the largest possible MSS, to support the most efficient use of compressed payloads. Unfortunately, some compression schemes occasionally need to transmit full headers (and thus smaller payloads) to resynchronize state at their endpoint compressors/decompressors. If the largest MTU is used to calculate the value to advertise in the MSS option, TCP retransmission may interfere with compressor resynchronization.

As a result, when the effective MTU of an interface varies packet-to-packet, TCP implementations SHOULD use the smallest effective MTU of the interface to calculate the value to advertise in the MSS option (SHLD-6).

3.6.4. Nagle Algorithm

The "Nagle algorithm" was described in RFC 896 [14] and was recommended in RFC 1122 [15] for mitigation of an early problem of too many small packets being generated. It has been implemented in most current TCP code bases, sometimes with minor variations (see Appendix A.3).

If there is unacknowledged data (i.e., SND.NXT > SND.UNA), then the sending TCP endpoint buffers all user data (regardless of the PSH bit), until the outstanding data has been acknowledged or until the TCP endpoint can send a full-sized segment (Eff.snd.MSS bytes).

A TCP implementation SHOULD implement the Nagle Algorithm to coalesce short segments (SHLD-7). However, there MUST be a way for an application to disable the Nagle algorithm on an individual connection (MUST-17). In all cases, sending data is also subject to the limitation imposed by the Slow Start algorithm [31].

Since there can be problematic interactions between the Nagle Algorithm and delayed acknowledgements, some implementations use minor variations of the Nagle algorithm, such as the one described in Appendix A.3.

3.6.5. IPv6 Jumbograms

In order to support TCP over IPv6 Jumbograms, implementations need to be able to send TCP segments larger than the 64KB limit that the MSS option can convey. RFC 2675 [6] defines that an MSS value of 65,535 bytes is to be treated as infinity, and Path MTU Discovery [3] is used to determine the actual MSS.

The Jumbo Payload option need not be implemented or understood by IPv6 nodes that do not support attachment to links with a MTU greater than 65,575 [6], and the present IPv6 Node Requirements does not include support for Jumbograms [50].

3.7. Data Communication

Once the connection is established data is communicated by the exchange of segments. Because segments may be lost due to errors (checksum test failure), or network congestion, TCP uses retransmission to ensure delivery of every segment. Duplicate segments may arrive due to network or TCP retransmission. As discussed in the section on sequence numbers the TCP implementation performs certain tests on the sequence and acknowledgment numbers in the segments to verify their acceptability.

The sender of data keeps track of the next sequence number to use in the variable SND.NXT. The receiver of data keeps track of the next sequence number to expect in the variable RCV.NXT. The sender of data keeps track of the oldest unacknowledged sequence number in the variable SND.UNA. If the data flow is momentarily idle and all data sent has been acknowledged then the three variables will be equal.

When the sender creates a segment and transmits it the sender advances SND.NXT. When the receiver accepts a segment it advances RCV.NXT and sends an acknowledgment. When the data sender receives an acknowledgment it advances SND.UNA. The extent to which the values of these variables differ is a measure of the delay in the communication. The amount by which the variables are advanced is the length of the data and SYN or FIN flags in the segment. Note that once in the ESTABLISHED state all segments must carry current acknowledgment information.

The CLOSE user call implies a push function, as does the FIN control flag in an incoming segment.

3.7.1. Retransmission Timeout

Because of the variability of the networks that compose an internetwork system and the wide range of uses of TCP connections the retransmission timeout (RTO) must be dynamically determined.

The RTO MUST be computed according to the algorithm in [9], including Karn's algorithm for taking RTT samples (MUST-18).

RFC 793 contains an early example procedure for computing the RTO. This was then replaced by the algorithm described in RFC 1122, and subsequently updated in RFC 2988, and then again in RFC 6298.

RFC 1122 allows that if a retransmitted packet is identical to the original packet (which implies not only that the data boundaries have not changed, but also that none of the headers have changed), then the same IPv4 Identification field MAY be used (see Section 3.2.1.5 of RFC 1122) (MAY-4). The same IP identification field may be reused anyways, since it is only meaningful when a datagram is fragmented [40]. TCP implementations should not rely on or typically interact with this IPv4 header field in any way. It is not a reasonable way to either indicate duplicate sent segments, nor to identify duplicate received segments.

3.7.2. TCP Congestion Control

RFC 1122 required implementation of Van Jacobson's congestion control algorithm combining slow start with congestion avoidance. RFC 2581 provided IETF Standards Track description of this, along with fast retransmit and fast recovery. RFC 5681 is the current description of these algorithms and is the current standard for TCP congestion control.

A TCP endpoint MUST implement RFC 5681 (MUST-19).

Explicit Congestion Notification (ECN) was defined in RFC 3168 and is an IETF Standards Track enhancement that has many benefits [47].

A TCP endpoint SHOULD implement ECN as described in RFC 3168 (SHLD-8).

3.7.3. TCP Connection Failures

Excessive retransmission of the same segment by a TCP endpoint indicates some failure of the remote host or the Internet path. This failure may be of short or long duration. The following procedure MUST be used to handle excessive retransmissions of data segments (MUST-20):

(a) There are two thresholds R1 and R2 measuring the amount of retransmission that has occurred for the same segment. R1 and R2 might be measured in time units or as a count of retransmissions.

(b) When the number of transmissions of the same segment reaches or exceeds threshold R1, pass negative advice (see Section 3.3.1.4 of [15]) to the IP layer, to trigger dead-gateway diagnosis.

(c) When the number of transmissions of the same segment reaches a threshold R2 greater than R1, close the connection.

(d) An application MUST (MUST-21) be able to set the value for R2 for a particular connection. For example, an interactive application might set R2 to "infinity," giving the user control over when to disconnect.

(e) TCP implementations SHOULD inform the application of the delivery problem (unless such information has been disabled by the application; see Asynchronous Reports section), when R1 is reached and before R2 (SHLD-9). This will allow a remote login (User Telnet) application program to inform the user, for example.

The value of R1 SHOULD correspond to at least 3 retransmissions, at the current RTO (SHLD-10). The value of R2 SHOULD correspond to at least 100 seconds (SHLD-11).

An attempt to open a TCP connection could fail with excessive retransmissions of the SYN segment or by receipt of a RST segment or an ICMP Port Unreachable. SYN retransmissions MUST be handled in the general way just described for data retransmissions, including notification of the application layer.

However, the values of R1 and R2 may be different for SYN and data segments. In particular, R2 for a SYN segment MUST be set large enough to provide retransmission of the segment for at least 3 minutes (MUST-23). The application can close the connection (i.e., give up on the open attempt) sooner, of course.

3.7.4. TCP Keep-Alives

Implementors MAY include "keep-alives" in their TCP implementations (MAY-5), although this practice is not universally accepted. Some TCP implementations, however, have included a keep-alive mechanism. To confirm that an idle connection is still active, these implementations send a probe segment designed to elicit a response from the TCP peer. Such a segment generally contains SEG.SEQ = SND.NXT-1 and may or may not contain one garbage octet of data. If keep-alives are included, the application MUST be able to turn them on or off for each TCP connection (MUST-24), and they MUST default to off (MUST-25).

Keep-alive packets MUST only be sent when no sent data is outstanding, and no data or acknowledgement packets have been received for the connection within an interval (MUST-26). This interval MUST be configurable (MUST-27) and MUST default to no less than two hours (MUST-28).

It is extremely important to remember that ACK segments that contain no data are not reliably transmitted by TCP. Consequently, if a keep-alive mechanism is implemented it MUST NOT interpret failure to respond to any specific probe as a dead connection (MUST-29).

An implementation SHOULD send a keep-alive segment with no data (SHLD-12); however, it MAY be configurable to send a keep-alive segment containing one garbage octet (MAY-6), for compatibility with erroneous TCP implementations.

3.7.5. The Communication of Urgent Information

As a result of implementation differences and middlebox interactions, new applications SHOULD NOT employ the TCP urgent mechanism (SHLD-13). However, TCP implementations MUST still include support for the urgent mechanism (MUST-30). Details can be found in RFC 6093 [35].

The objective of the TCP urgent mechanism is to allow the sending user to stimulate the receiving user to accept some urgent data and to permit the receiving TCP endpoint to indicate to the receiving user when all the currently known urgent data has been received by the user.

This mechanism permits a point in the data stream to be designated as the end of urgent information. Whenever this point is in advance of the receive sequence number (RCV.NXT) at the receiving TCP endpoint, that TCP must tell the user to go into "urgent mode"; when the receive sequence number catches up to the urgent pointer, the TCP implementation must tell user to go into "normal mode". If the urgent pointer is updated while the user is in "urgent mode", the update will be invisible to the user.

The method employs an urgent field that is carried in all segments transmitted. The URG control flag indicates that the urgent field is meaningful and must be added to the segment sequence number to yield the urgent pointer. The absence of this flag indicates that there is no urgent data outstanding.

To send an urgent indication the user must also send at least one data octet. If the sending user also indicates a push, timely delivery of the urgent information to the destination process is enhanced.

A TCP implementation MUST support a sequence of urgent data of any length (MUST-31). [15]

The urgent pointer MUST point to the sequence number of the octet following the urgent data (MUST-62).

A TCP implementation MUST (MUST-32) inform the application layer asynchronously whenever it receives an Urgent pointer and there was previously no pending urgent data, or whenever the Urgent pointer advances in the data stream. The TCP implementation MUST (MUST-33) provide a way for the application to learn how much urgent data remains to be read from the connection, or at least to determine whether or not more urgent data remains to be read [15].

3.7.6. Managing the Window

The window sent in each segment indicates the range of sequence numbers the sender of the window (the data receiver) is currently prepared to accept. There is an assumption that this is related to the currently available data buffer space available for this connection.

The sending TCP endpoint packages the data to be transmitted into segments that fit the current window, and may repackage segments on the retransmission queue. Such repackaging is not required, but may be helpful.

In a connection with a one-way data flow, the window information will be carried in acknowledgment segments that all have the same sequence number so there will be no way to reorder them if they arrive out of order. This is not a serious problem, but it will allow the window information to be on occasion temporarily based on old reports from the data receiver. A refinement to avoid this problem is to act on the window information from segments that carry the highest acknowledgment number (that is segments with acknowledgment number equal or greater than the highest previously received).

Indicating a large window encourages transmissions. If more data arrives than can be accepted, it will be discarded. This will result in excessive retransmissions, adding unnecessarily to the load on the network and the TCP endpoints. Indicating a small window may restrict the transmission of data to the point of introducing a round trip delay between each new segment transmitted.

The mechanisms provided allow a TCP endpoint to advertise a large window and to subsequently advertise a much smaller window without having accepted that much data. This, so called "shrinking the window," is strongly discouraged. The robustness principle [15] dictates that TCP peers will not shrink the window themselves, but will be prepared for such behavior on the part of other TCP peers.

A TCP receiver SHOULD NOT shrink the window, i.e., move the right window edge to the left (SHLD-14). However, a sending TCP peer MUST be robust against window shrinking, which may cause the "useable window" (see Section 3.7.6.2.1) to become negative (MUST-34).

If this happens, the sender SHOULD NOT send new data (SHLD-15), but SHOULD retransmit normally the old unacknowledged data between SND.UNA and SND.UNA+SND.WND (SHLD-16). The sender MAY also retransmit old data beyond SND.UNA+SND.WND (MAY-7), but SHOULD NOT time out the connection if data beyond the right window edge is not acknowledged (SHLD-17). If the window shrinks to zero, the TCP implementation MUST probe it in the standard way (described below) (MUST-35).

3.7.6.1. Zero Window Probing

The sending TCP peer must be prepared to accept from the user and send at least one octet of new data even if the send window is zero. The sending TCP peer must regularly retransmit to the receiving TCP peer even when the window is zero, in order to "probe" the window. Two minutes is recommended for the retransmission interval when the window is zero. This retransmission is essential to guarantee that when either TCP peer has a zero window the re-opening of the window will be reliably reported to the other. This is referred to as Zero-Window Probing (ZWP) in other documents.

Probing of zero (offered) windows MUST be supported (MUST-36).

A TCP implementation MAY keep its offered receive window closed indefinitely (MAY-8). As long as the receiving TCP peer continues to send acknowledgments in response to the probe segments, the sending TCP peer MUST allow the connection to stay open (MUST-37). This enables TCP to function in scenarios such as the "printer ran out of paper" situation described in Section 4.2.2.17 of RFC1122. The behavior is subject to the implementation’s resource management concerns, as noted in [37].

When the receiving TCP peer has a zero window and a segment arrives it must still send an acknowledgment showing its next expected sequence number and current window (zero).

The transmitting host SHOULD send the first zero-window probe when a zero window has existed for the retransmission timeout period (SHLD-29) (Section 3.7.1), and SHOULD increase exponentially the interval between successive probes (SHLD-30).

3.7.6.2. Silly Window Syndrome Avoidance

The "Silly Window Syndrome" (SWS) is a stable pattern of small incremental window movements resulting in extremely poor TCP performance. Algorithms to avoid SWS are described below for both the sending side and the receiving side. RFC 1122 contains more detailed discussion of the SWS problem. Note that the Nagle algorithm and the sender SWS avoidance algorithm play complementary roles in improving performance. The Nagle algorithm discourages sending tiny segments when the data to be sent increases in small increments, while the SWS avoidance algorithm discourages small segments resulting from the right window edge advancing in small increments.

3.7.6.2.1. Sender’s Algorithm - When to Send Data

A TCP implementation MUST include a SWS avoidance algorithm in the sender (MUST-38).

The Nagle algorithm from Section 3.6.4 additionally describes how to coalesce short segments.

The sender’s SWS avoidance algorithm is more difficult than the receivers’s, because the sender does not know (directly) the receiver’s total buffer space RCV.BUFF. An approach that has been found to work well is for the sender to calculate Max(SND.WND), the maximum send window it has seen so far on the connection, and to use this value as an estimate of RCV.BUFF. Unfortunately, this can only be an estimate; the receiver may at any time reduce the size of RCV.BUFF. To avoid a resulting deadlock, it is necessary to have a timeout to force transmission of data, overriding the SWS avoidance algorithm. In practice, this timeout should seldom occur.

The "useable window" is:

```
U = SND.UNA + SND.WND - SND.NXT
```

i.e., the offered window less the amount of data sent but not acknowledged. If D is the amount of data queued in the sending TCP endpoint but not yet sent, then the following set of rules is recommended.

Send data:

(1) if a maximum-sized segment can be sent, i.e, if:

```
min(D,U) >= Eff.snd.MSS;
```

(2) or if the data is pushed and all queued data can be sent now, i.e., if:

```
[SND.NXT = SND.UNA and] PUSHED and D <= U
```

(the bracketed condition is imposed by the Nagle algorithm);

(3) or if at least a fraction Fs of the maximum window can be sent, i.e., if:

```
[SND.NXT = SND.UNA and]

   min(D.U) >= Fs * Max(SND.WND);
```

(4) or if data is PUSHed and the override timeout occurs.

Here Fs is a fraction whose recommended value is 1/2. The override timeout should be in the range 0.1 - 1.0 seconds. It may be convenient to combine this timer with the timer used to probe zero windows (Section 3.7.6.1).

3.7.6.2.2. Receiver’s Algorithm - When to Send a Window Update

A TCP implementation MUST include a SWS avoidance algorithm in the receiver (MUST-39).

The receiver’s SWS avoidance algorithm determines when the right window edge may be advanced; this is customarily known as "updating the window". This algorithm combines with the delayed ACK algorithm (Section 3.7.6.3) to determine when an ACK segment containing the current window will really be sent to the receiver.

The solution to receiver SWS is to avoid advancing the right window edge RCV.NXT+RCV.WND in small increments, even if data is received from the network in small segments.

Suppose the total receive buffer space is RCV.BUFF. At any given moment, RCV.USER octets of this total may be tied up with data that has been received and acknowledged but that the user process has not yet consumed. When the connection is quiescent, RCV.WND = RCV.BUFF and RCV.USER = 0.

Keeping the right window edge fixed as data arrives and is acknowledged requires that the receiver offer less than its full buffer space, i.e., the receiver must specify a RCV.WND that keeps RCV.NXT+RCV.WND constant as RCV.NXT increases. Thus, the total buffer space RCV.BUFF is generally divided into three parts:

```
             |<------- RCV.BUFF ---------------->|
                  1             2            3
         ----|---------|------------------|------|----
                    RCV.NXT               ^
                                       (Fixed)

         1 - RCV.USER =  data received but not yet consumed;
         2 - RCV.WND =   space advertised to sender;
         3 - Reduction = space available but not yet
                         advertised.
```

The suggested SWS avoidance algorithm for the receiver is to keep RCV.NXT+RCV.WND fixed until the reduction satisfies:

$$\mathrm{RCV.BUFF} - \mathrm{RCV.USER} - \mathrm{RCV.WND} >= \min(\ \mathrm{Fr} * \mathrm{RCV.BUFF},\ \mathrm{Eff.snd.MSS}\ )$$

where Fr is a fraction whose recommended value is 1/2, and Eff.snd.MSS is the effective send MSS for the connection (see Section 3.6.1). When the inequality is satisfied, RCV.WND is set to RCV.BUFF-RCV.USER.

Note that the general effect of this algorithm is to advance RCV.WND in increments of Eff.snd.MSS (for realistic receive buffers: Eff.snd.MSS < RCV.BUFF/2). Note also that the receiver must use its own Eff.snd.MSS, assuming it is the same as the sender's.

3.7.6.3. Delayed Acknowledgements - When to Send an ACK Segment

A host that is receiving a stream of TCP data segments can increase efficiency in both the Internet and the hosts by sending fewer than one ACK (acknowledgment) segment per data segment received; this is known as a "delayed ACK".

A TCP endpoint SHOULD implement a delayed ACK (SHLD-18), but an ACK should not be excessively delayed; in particular, the delay MUST be less than 0.5 seconds (MUST-40), and in a stream of full-sized segments there SHOULD be an ACK for at least every second segment (SHLD-19). Excessive delays on ACKs can disturb the round-trip timing and packet "clocking" algorithms. More complete discussion of delayed ACK behavior is in Section 4.2 of RFC 5681 [31], including rules for streams of segments that are not full-sized. Note that there are several current practices that further lead to a reduced number of ACKs, including generic receive offload (GRO), ACK compression, and ACK decimation [22].

3.8. Interfaces

There are of course two interfaces of concern: the user/TCP interface and the TCP/lower-level interface. We have a fairly elaborate model of the user/TCP interface, but the interface to the lower level protocol module is left unspecified here, since it will be specified in detail by the specification of the lower level protocol. For the case that the lower level is IP we note some of the parameter values that TCP implementations might use.

3.8.1. User/TCP Interface

The following functional description of user commands to the TCP implementation is, at best, fictional, since every operating system will have different facilities. Consequently, we must warn readers that different TCP implementations may have different user interfaces. However, all TCP implementations must provide a certain minimum set of services to guarantee that all TCP implementations can support the same protocol hierarchy. This section specifies the functional interfaces required of all TCP implementations.

Section 3.1 of [49] also identifies primitives provided by TCP, and could be used as an additional reference for implementers.

TCP User Commands

The following sections functionally characterize a USER/TCP interface. The notation used is similar to most procedure or function calls in high level languages, but this usage is not meant to rule out trap type service calls.

The user commands described below specify the basic functions the TCP implementation must perform to support interprocess communication. Individual implementations must define their own exact format, and may provide combinations or subsets of the basic functions in single calls. In particular, some implementations may wish to automatically OPEN a connection on the first SEND or RECEIVE issued by the user for a given connection.

In providing interprocess communication facilities, the TCP implementation must not only accept commands, but must also return information to the processes it serves. The latter consists of:

(a) general information about a connection (e.g., interrupts, remote close, binding of unspecified remote socket).

(b) replies to specific user commands indicating success or various types of failure.

Open

Format: OPEN (local port, remote socket, active/passive [, timeout] [, DiffServ field] [, security/compartment] [local IP address,] [, options]) -> local connection name

If the active/passive flag is set to passive, then this is a call to LISTEN for an incoming connection. A passive open may have either a fully specified remote socket to wait for a particular connection or an unspecified remote socket to wait for any call. A fully specified passive call can be made active by the subsequent execution of a SEND.

A transmission control block (TCB) is created and partially filled in with data from the OPEN command parameters.

Every passive OPEN call either creates a new connection record in LISTEN state, or it returns an error; it MUST NOT affect any previously created connection record (MUST-41).

A TCP implementation that supports multiple concurrent connections MUST provide an OPEN call that will functionally allow an application to LISTEN on a port while a connection block with the same local port is in SYN-SENT or SYN-RECEIVED state (MUST-42).

On an active OPEN command, the TCP endpoint will begin the procedure to synchronize (i.e., establish) the connection at once.

The timeout, if present, permits the caller to set up a timeout for all data submitted to TCP. If data is not successfully delivered to the destination within the timeout period, the TCP endpoint will abort the connection. The present global default is five minutes.

The TCP implementation or some component of the operating system will verify the users authority to open a connection with the specified DiffServ field value or security/compartment. The absence of a DiffServ field value or security/compartment specification in the OPEN call indicates the default values must be used.

TCP will accept incoming requests as matching only if the security/compartment information is exactly the same as that requested in the OPEN call.

The DiffServ field value indicated by the user only impacts outgoing packets, may be altered en route through the network, and has no direct bearing or relation to received packets.

A local connection name will be returned to the user by the TCP implementation. The local connection name can then be used as a short hand term for the connection defined by the <local socket, remote socket> pair.

The optional "local IP address" parameter MUST be supported to allow the specification of the local IP address (MUST-43). This enables applications that need to select the local IP address used when multihoming is present.

A passive OPEN call with a specified "local IP address" parameter will await an incoming connection request to that address. If the parameter is unspecified, a passive OPEN will await an incoming connection request to any local IP address, and then bind the local IP address of the connection to the particular address that is used.

For an active OPEN call, a specified "local IP address" parameter will be used for opening the connection. If the parameter is unspecified, the host will choose an appropriate local IP address (see RFC 1122 section 3.3.4.2).

If an application on a multihomed host does not specify the local IP address when actively opening a TCP connection, then the TCP implementation MUST ask the IP layer to select a local IP address before sending the (first) SYN (MUST-44). See the function GET_SRCADDR() in Section 3.4 of RFC 1122.

At all other times, a previous segment has either been sent or received on this connection, and TCP implementations MUST use the same local address is used that was used in those previous segments (MUST-45).

A TCP implementation MUST reject as an error a local OPEN call for an invalid remote IP address (e.g., a broadcast or multicast address) (MUST-46).

Send

Format: SEND (local connection name, buffer address, byte count, PUSH flag (optional), URGENT flag [,timeout])

This call causes the data contained in the indicated user buffer to be sent on the indicated connection. If the connection has not been opened, the SEND is considered an error. Some implementations may allow users to SEND first; in which case, an automatic OPEN would be done. For example, this might be one way for application data to be included in SYN segments. If the calling process is not authorized to use this connection, an error is returned.

A TCP endpoint MAY implement PUSH flags on SEND calls (MAY-15). If PUSH flags are not implemented, then the sending TCP peer:

(1) MUST NOT buffer data indefinitely (MUST-60), and (2) MUST set the PSH bit in the last buffered segment (i.e., when there is no more queued data to be sent) (MUST-61). The remaining description below assumes the PUSH flag is supported on SEND calls.

If the PUSH flag is set, the application intends the data to be transmitted promptly to the receiver, and the PUSH bit will be set in the last TCP segment created from the buffer. When an application issues a series of SEND calls without setting the PUSH flag, the TCP implementation MAY aggregate the data internally without sending it (MAY-16).

The PSH bit is not a record marker and is independent of segment boundaries. The transmitter SHOULD collapse successive bits when it packetizes data, to send the largest possible segment (SHLD-27).

If the PUSH flag is not set, the data may be combined with data from subsequent SENDs for transmission efficiency. Note that when the Nagle algorithm is in use, TCP implementations may buffer the data before sending, without regard to the PUSH flag (see Section 3.6.4).

An application program is logically required to set the PUSH flag in a SEND call whenever it needs to force delivery of the data to avoid a communication deadlock. However, a TCP implementation SHOULD send a maximum-sized segment whenever possible (SHLD-28), to improve performance (see Section 3.7.6.2.1).

New applications SHOULD NOT set the URGENT flag [35] due to implementation differences and middlebox issues (SHLD-13).

If the URGENT flag is set, segments sent to the destination TCP peer will have the urgent pointer set. The receiving TCP peer will signal the urgent condition to the receiving process if the urgent pointer indicates that data preceding the urgent pointer has not been consumed by the receiving process. The purpose of urgent is to stimulate the receiver to process the urgent data and to indicate to the receiver when all the currently known urgent data has been received. The number of times the sending user's TCP implementation signals urgent will not necessarily be equal to the number of times the receiving user will be notified of the presence of urgent data.

If no remote socket was specified in the OPEN, but the connection is established (e.g., because a LISTENing connection has become specific due to a remote segment arriving for the local socket), then the designated buffer is sent to the implied remote socket. Users who make use of OPEN with an unspecified remote socket can make use of SEND without ever explicitly knowing the remote socket address.

However, if a SEND is attempted before the remote socket becomes specified, an error will be returned. Users can use the STATUS call to determine the status of the connection. Some TCP implementations may notify the user when an unspecified socket is bound.

If a timeout is specified, the current user timeout for this connection is changed to the new one.

In the simplest implementation, SEND would not return control to the sending process until either the transmission was complete or the timeout had been exceeded. However, this simple method is both subject to deadlocks (for example, both sides of the connection might try to do SENDs before doing any RECEIVEs) and offers poor performance, so it is not recommended. A more sophisticated implementation would return immediately to allow the process to run concurrently with network I/O, and, furthermore, to allow multiple SENDs to be in progress. Multiple SENDs are served in first come, first served order, so the TCP endpoint will queue those it cannot service immediately.

We have implicitly assumed an asynchronous user interface in which a SEND later elicits some kind of SIGNAL or pseudo-interrupt from the serving TCP endpoint. An alternative is to return a response immediately. For instance, SENDs might return immediate local acknowledgment, even if the segment sent had not been acknowledged by the distant TCP endpoint. We could optimistically assume eventual success. If we are wrong, the connection will close anyway due to the timeout. In implementations of this kind (synchronous), there will still be some asynchronous signals, but these will deal with the connection itself, and not with specific segments or buffers.

In order for the process to distinguish among error or success indications for different SENDs, it might be appropriate for the buffer address to be returned along with the coded response to the SEND request. TCP-to-user signals are discussed below, indicating the information that should be returned to the calling process.

Receive

Format: RECEIVE (local connection name, buffer address, byte count) -> byte count, urgent flag, push flag (optional)

This command allocates a receiving buffer associated with the specified connection. If no OPEN precedes this command or the calling process is not authorized to use this connection, an error is returned.

In the simplest implementation, control would not return to the calling program until either the buffer was filled, or some error occurred, but this scheme is highly subject to deadlocks. A more sophisticated implementation would permit several RECEIVEs to be outstanding at once. These would be filled as segments arrive. This strategy permits increased throughput at the cost of a more elaborate scheme (possibly asynchronous) to notify the calling program that a PUSH has been seen or a buffer filled.

A TCP receiver MAY pass a received PSH flag to the application layer via the PUSH flag in the interface (MAY-17), but it is not required (this was clarified in RFC 1122 section 4.2.2.2). The remainder of text describing the RECEIVE call below assumes that passing the PUSH indication is supported.

If enough data arrive to fill the buffer before a PUSH is seen, the PUSH flag will not be set in the response to the RECEIVE. The buffer will be filled with as much data as it can hold. If a PUSH is seen before the buffer is filled the buffer will be returned partially filled and PUSH indicated.

If there is urgent data the user will have been informed as soon as it arrived via a TCP-to-user signal. The receiving user should thus be in "urgent mode". If the URGENT flag is on, additional urgent data remains. If the URGENT flag is off, this call to RECEIVE has returned all the urgent data, and the user may now leave "urgent mode". Note that data following the urgent pointer (non-urgent data) cannot be delivered to the user in the same buffer with preceding urgent data unless the boundary is clearly marked for the user.

To distinguish among several outstanding RECEIVEs and to take care of the case that a buffer is not completely filled, the return code is accompanied by both a buffer pointer and a byte count indicating the actual length of the data received.

Alternative implementations of RECEIVE might have the TCP endpoint allocate buffer storage, or the TCP endpoint might share a ring buffer with the user.

Close

Format: CLOSE (local connection name)

This command causes the connection specified to be closed. If the connection is not open or the calling process is not authorized to use this connection, an error is returned. Closing connections is intended to be a graceful operation in the sense that outstanding SENDs will be transmitted (and retransmitted), as flow control permits, until all have been serviced. Thus, it should be acceptable to make several SEND calls, followed by a CLOSE, and expect all the data to be sent to the destination. It should also be clear that users should continue to RECEIVE on CLOSING connections, since the remote peer may be trying to transmit the last of its data. Thus, CLOSE means "I have no more to send" but does not mean "I will not receive any more." It may happen (if the user level protocol is not well thought out) that the closing side is unable to get rid of all its data before timing out. In this event, CLOSE turns into ABORT, and the closing TCP peer gives up.

The user may CLOSE the connection at any time on their own initiative, or in response to various prompts from the TCP implementation (e.g., remote close executed, transmission timeout exceeded, destination inaccessible).

Because closing a connection requires communication with the remote TCP peer, connections may remain in the closing state for a short time. Attempts to reopen the connection before the TCP peer replies to the CLOSE command will result in error responses.

Close also implies push function.

Status

Format: STATUS (local connection name) -> status data

This is an implementation dependent user command and could be excluded without adverse effect. Information returned would typically come from the TCB associated with the connection.

This command returns a data block containing the following information:

local socket,
remote socket,

local connection name,
receive window,
send window,
connection state,
number of buffers awaiting acknowledgment,
number of buffers pending receipt,
urgent state,
DiffServ field value,
security/compartment,
and transmission timeout.

Depending on the state of the connection, or on the implementation itself, some of this information may not be available or meaningful. If the calling process is not authorized to use this connection, an error is returned. This prevents unauthorized processes from gaining information about a connection.

Abort

Format: ABORT (local connection name)

This command causes all pending SENDs and RECEIVES to be aborted, the TCB to be removed, and a special RESET message to be sent to the remote TCP peer of the connection. Depending on the implementation, users may receive abort indications for each outstanding SEND or RECEIVE, or may simply receive an ABORT-acknowledgment.

Flush

Some TCP implementations have included a FLUSH call, which will empty the TCP send queue of any data that the user has issued SEND calls but is still to the right of the current send window. That is, it flushes as much queued send data as possible without losing sequence number synchronization. The FLUSH call MAY be implemented (MAY-14).

Asynchronous Reports

There MUST be a mechanism for reporting soft TCP error conditions to the application (MUST-47). Generically, we assume this takes the form of an application-supplied ERROR_REPORT routine that may be upcalled asynchronously from the transport layer:

```
ERROR_REPORT(local connection name, reason, subreason)
```

The precise encoding of the reason and subreason parameters is not specified here. However, the conditions that are reported asynchronously to the application MUST include:

- * ICMP error message arrived (see Section 3.8.2.2 for description of handling each ICMP message type, since some message types need to be suppressed from generating reports to the application)

- * Excessive retransmissions (see Section 3.7.3)

- * Urgent pointer advance (see Section 3.7.5)

However, an application program that does not want to receive such ERROR_REPORT calls SHOULD be able to effectively disable these calls (SHLD-20).

Set Differentiated Services Field (IPv4 TOS or IPv6 Traffic Class)

The application layer MUST be able to specify the Differentiated Services field for segments that are sent on a connection (MUST-48). The Differentiated Services field includes the 6-bit Differentiated Services Code Point (DSCP) value. It is not required, but the application SHOULD be able to change the Differentiated Services field during the connection lifetime (SHLD-21). TCP implementations SHOULD pass the current Differentiated Services field value without change to the IP layer, when it sends segments on the connection (SHLD-22).

The Differentiated Services field will be specified independently in each direction on the connection, so that the receiver application will specify the Differentiated Services field used for ACK segments.

TCP implementations MAY pass the most recently received Differentiated Services field up to the application (MAY-9).

3.8.2. TCP/Lower-Level Interface

The TCP endpoint calls on a lower level protocol module to actually send and receive information over a network. The two current standard Internet Protocol (IP) versions layered below TCP are IPv4 [1] and IPv6 [12].

If the lower level protocol is IPv4 it provides arguments for a type of service (used within the Differentiated Services field) and for a time to live. TCP uses the following settings for these parameters:

DiffServ field: The IP header value for the DiffServ field is given by the user. This includes the bits of the DiffServ Code Point (DSCP).

Time to Live (TTL): The TTL value used to send TCP segments MUST be configurable (MUST-49).

Note that RFC 793 specified one minute (60 seconds) as a constant for the TTL, because the assumed maximum segment lifetime was two minutes. This was intended to explicitly ask that a segment be destroyed if it cannot be delivered by the internet system within one minute. RFC 1122 changed this specification to require that the TTL be configurable.

Note that the DiffServ field is permitted to change during a connection (Section 4.2.4.2 of RFC 1122). However, the application interface might not support this ability, and the application does not have knowledge about individual TCP segments, so this can only be done on a coarse granularity, at best. This limitation is further discussed in RFC 7657 (sec 5.1, 5.3, and 6) [46]. Generally, an application SHOULD NOT change the DiffServ field value during the course of a connection (SHLD-23).

Any lower level protocol will have to provide the source address, destination address, and protocol fields, and some way to determine the "TCP length", both to provide the functional equivalent service of IP and to be used in the TCP checksum.

When received options are passed up to TCP from the IP layer, TCP implementations MUST ignore options that it does not understand (MUST-50).

A TCP implementation MAY support the Time Stamp (MAY-10) and Record Route (MAY-11) options.

3.8.2.1. Source Routing

If the lower level is IP (or other protocol that provides this feature) and source routing is used, the interface must allow the route information to be communicated. This is especially important so that the source and destination addresses used in the TCP checksum be the originating source and ultimate destination. It is also important to preserve the return route to answer connection requests.

An application MUST be able to specify a source route when it actively opens a TCP connection (MUST-51), and this MUST take precedence over a source route received in a datagram (MUST-52).

When a TCP connection is OPENed passively and a packet arrives with a completed IP Source Route option (containing a return route), TCP implementations MUST save the return route and use it for all segments sent on this connection (MUST-53). If a different source route arrives in a later segment, the later definition SHOULD override the earlier one (SHLD-24).

3.8.2.2. ICMP Messages

TCP implementations MUST act on an ICMP error message passed up from the IP layer, directing it to the connection that created the error (MUST-54). The necessary demultiplexing information can be found in the IP header contained within the ICMP message.

This applies to ICMPv6 in addition to IPv4 ICMP.

[29] contains discussion of specific ICMP and ICMPv6 messages classified as either "soft" or "hard" errors that may bear different responses. Treatment for classes of ICMP messages is described below:

Source Quench
TCP implementations MUST silently discard any received ICMP Source Quench messages (MUST-55). See [10] for discussion.

Soft Errors
For ICMP these include: Destination Unreachable -- codes 0, 1, 5, Time Exceeded -- codes 0, 1, and Parameter Problem.
For ICMPv6 these include: Destination Unreachable -- codes 0 and 3, Time Exceeded -- codes 0, 1, and Parameter Problem -- codes 0, 1, 2
Since these Unreachable messages indicate soft error conditions, TCP implementations MUST NOT abort the connection (MUST-56), and it SHOULD make the information available to the application (SHLD-25).

Hard Errors
For ICMP these include Destination Unreachable -- codes 2-4">
These are hard error conditions, so TCP implementations SHOULD abort the connection (SHLD-26). [29] notes that some implementations do not abort connections when an ICMP hard error is received for a connection that is in any of the synchronized states.

Note that [29] section 4 describes widespread implementation behavior that treats soft errors as hard errors during connection establishment.

3.8.2.3. Source Address Validation

RFC 1122 requires addresses to be validated in incoming SYN packets:

> An incoming SYN with an invalid source address MUST be ignored either by TCP or by the IP layer (MUST-63) (Section 3.2.1.3 of [15]).
>
> A TCP implementation MUST silently discard an incoming SYN segment that is addressed to a broadcast or multicast address (MUST-57).

This prevents connection state and replies from being erroneously generated, and implementers should note that this guidance is applicable to all incoming segments, not just SYNs, as specifically indicated in RFC 1122.

3.9. Event Processing

The processing depicted in this section is an example of one possible implementation. Other implementations may have slightly different processing sequences, but they should differ from those in this section only in detail, not in substance.

The activity of the TCP endpoint can be characterized as responding to events. The events that occur can be cast into three categories: user calls, arriving segments, and timeouts. This section describes the processing the TCP endpoint does in response to each of the events. In many cases the processing required depends on the state of the connection.

Events that occur:

```
   User Calls

      OPEN
      SEND
      RECEIVE
      CLOSE
      ABORT
      STATUS

   Arriving Segments

      SEGMENT ARRIVES

   Timeouts

      USER TIMEOUT
```

RETRANSMISSION TIMEOUT
TIME-WAIT TIMEOUT

The model of the TCP/user interface is that user commands receive an immediate return and possibly a delayed response via an event or pseudo interrupt. In the following descriptions, the term "signal" means cause a delayed response.

Error responses in this document are identified by character strings. For example, user commands referencing connections that do not exist receive "error: connection not open".

Please note in the following that all arithmetic on sequence numbers, acknowledgment numbers, windows, et cetera, is modulo 2**32 the size of the sequence number space. Also note that "=<" means less than or equal to (modulo 2**32).

A natural way to think about processing incoming segments is to imagine that they are first tested for proper sequence number (i.e., that their contents lie in the range of the expected "receive window" in the sequence number space) and then that they are generally queued and processed in sequence number order.

When a segment overlaps other already received segments we reconstruct the segment to contain just the new data, and adjust the header fields to be consistent.

Note that if no state change is mentioned the TCP connection stays in the same state.

OPEN Call

CLOSED STATE (i.e., TCB does not exist)

Create a new transmission control block (TCB) to hold connection state information. Fill in local socket identifier, remote socket, DiffServ field, security/compartment, and user timeout information. Note that some parts of the remote socket may be unspecified in a passive OPEN and are to be filled in by the parameters of the incoming SYN segment. Verify the security and DiffServ value requested are allowed for this user, if not return "error: precedence not allowed" or "error: security/compartment not allowed." If passive enter the LISTEN state and return. If active and the remote socket is unspecified, return "error: remote socket unspecified"; if active and the remote socket is specified, issue a SYN segment. An initial send sequence number (ISS) is selected. A SYN segment of the form <SEQ=ISS><CTL=SYN> is sent. Set SND.UNA to ISS, SND.NXT to ISS+1, enter SYN-SENT state, and return.

If the caller does not have access to the local socket specified, return "error: connection illegal for this process". If there is no room to create a new connection, return "error: insufficient resources".

LISTEN STATE

If active and the remote socket is specified, then change the connection from passive to active, select an ISS. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: remote socket unspecified".

SYN-SENT STATE
SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection already exists".

SEND Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, then return "error: connection illegal for this process".

Otherwise, return "error: connection does not exist".

LISTEN STATE

If the remote socket is specified, then change the connection from passive to active, select an ISS. Send a SYN segment, set SND.UNA to ISS, SND.NXT to ISS+1. Enter SYN-SENT state. Data associated with SEND may be sent with SYN segment or queued for transmission after entering ESTABLISHED state. The urgent bit if requested in the command must be sent with the data segments sent as a result of this command. If there is no room to queue the request, respond with "error: insufficient resources". If Foreign socket was not specified, then return "error: remote socket unspecified".

SYN-SENT STATE
SYN-RECEIVED STATE

Queue the data for transmission after entering ESTABLISHED state. If no space to queue, respond with "error: insufficient resources".

ESTABLISHED STATE
CLOSE-WAIT STATE

Segmentize the buffer and send it with a piggybacked acknowledgment (acknowledgment value = RCV.NXT). If there is insufficient space to remember this buffer, simply return "error: insufficient resources".

If the urgent flag is set, then SND.UP <- SND.NXT and set the urgent pointer in the outgoing segments.

FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Return "error: connection closing" and do not service request.

RECEIVE Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE
SYN-SENT STATE
SYN-RECEIVED STATE

Queue for processing after entering ESTABLISHED state. If there is no room to queue this request, respond with "error: insufficient resources".

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

If insufficient incoming segments are queued to satisfy the request, queue the request. If there is no queue space to remember the RECEIVE, respond with "error: insufficient resources".

Reassemble queued incoming segments into receive buffer and return to user. Mark "push seen" (PUSH) if this is the case.

If RCV.UP is in advance of the data currently being passed to the user notify the user of the presence of urgent data.

When the TCP endpoint takes responsibility for delivering data to the user that fact must be communicated to the sender via an acknowledgment. The formation of such an acknowledgment is described below in the discussion of processing an incoming segment.

CLOSE-WAIT STATE

Since the remote side has already sent FIN, RECEIVEs must be satisfied by text already on hand, but not yet delivered to the user. If no text is awaiting delivery, the RECEIVE will get a "error: connection closing" response. Otherwise, any remaining text can be used to satisfy the RECEIVE.

CLOSING STATE
LAST-ACK STATE

TIME-WAIT STATE

Return "error: connection closing".

CLOSE Call

CLOSED STATE (i.e., TCB does not exist)

If the user does not have access to such a connection, return "error: connection illegal for this process".

Otherwise, return "error: connection does not exist".

LISTEN STATE

Any outstanding RECEIVEs are returned with "error: closing" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

Delete the TCB and return "error: closing" responses to any queued SENDs, or RECEIVEs.

SYN-RECEIVED STATE

If no SENDs have been issued and there is no pending data to send, then form a FIN segment and send it, and enter FIN-WAIT-1 state; otherwise queue for processing after entering ESTABLISHED state.

ESTABLISHED STATE

Queue this until all preceding SENDs have been segmentized, then form a FIN segment and send it. In any case, enter FIN-WAIT-1 state.

FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

Strictly speaking, this is an error and should receive a "error: connection closing" response. An "ok" response would be acceptable, too, as long as a second FIN is not emitted (the first FIN may be retransmitted though).

CLOSE-WAIT STATE

Queue this request until all preceding SENDs have been segmentized; then send a FIN segment, enter LAST-ACK state.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Respond with "error: connection closing".

ABORT Call

CLOSED STATE (i.e., TCB does not exist)

If the user should not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Any outstanding RECEIVEs should be returned with "error: connection reset" responses. Delete TCB, enter CLOSED state, and return.

SYN-SENT STATE

All queued SENDs and RECEIVEs should be given "connection reset" notification, delete the TCB, enter CLOSED state, and return.

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE

Send a reset segment:

```
<SEQ=SND.NXT><CTL=RST>
```

All queued SENDs and RECEIVEs should be given "connection reset" notification; all segments queued for transmission (except for the RST formed above) or retransmission should be flushed, delete the TCB, enter CLOSED state, and return.

CLOSING STATE LAST-ACK STATE TIME-WAIT STATE

Respond with "ok" and delete the TCB, enter CLOSED state, and return.

STATUS Call

CLOSED STATE (i.e., TCB does not exist)

If the user should not have access to such a connection, return "error: connection illegal for this process".

Otherwise return "error: connection does not exist".

LISTEN STATE

Return "state = LISTEN", and the TCB pointer.

SYN-SENT STATE

Return "state = SYN-SENT", and the TCB pointer.

SYN-RECEIVED STATE

Return "state = SYN-RECEIVED", and the TCB pointer.

ESTABLISHED STATE

Return "state = ESTABLISHED", and the TCB pointer.

FIN-WAIT-1 STATE

Return "state = FIN-WAIT-1", and the TCB pointer.

FIN-WAIT-2 STATE

Return "state = FIN-WAIT-2", and the TCB pointer.

CLOSE-WAIT STATE

Return "state = CLOSE-WAIT", and the TCB pointer.

CLOSING STATE

Return "state = CLOSING", and the TCB pointer.

LAST-ACK STATE

Return "state = LAST-ACK", and the TCB pointer.

TIME-WAIT STATE

Return "state = TIME-WAIT", and the TCB pointer.

SEGMENT ARRIVES

If the state is CLOSED (i.e., TCB does not exist) then

all data in the incoming segment is discarded. An incoming segment containing a RST is discarded. An incoming segment not containing a RST causes a RST to be sent in response. The acknowledgment and sequence field values are selected to make the reset sequence acceptable to the TCP endpoint that sent the offending segment.

If the ACK bit is off, sequence number zero is used,

```
<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>
```

If the ACK bit is on,

```
<SEQ=SEG.ACK><CTL=RST>
```

Return.

If the state is LISTEN then

first check for an RST

An incoming RST should be ignored. Return.

second check for an ACK

Any acknowledgment is bad if it arrives on a connection still in the LISTEN state. An acceptable reset segment should be formed for any arriving ACK-bearing segment. The RST should be formatted as follows:

```
<SEQ=SEG.ACK><CTL=RST>
```

Return.

third check for a SYN

If the SYN bit is set, check the security. If the security/compartment on the incoming segment does not exactly match the security/compartment in the TCB then send a reset and return.

```
<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>
```

Set RCV.NXT to SEG.SEQ+1, IRS is set to SEG.SEQ and any other control or text should be queued for processing later. ISS should be selected and a SYN segment sent of the form:

<SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>

SND.NXT is set to ISS+1 and SND.UNA to ISS. The connection state should be changed to SYN-RECEIVED. Note that any other incoming control or data (combined with SYN) will be processed in the SYN-RECEIVED state, but processing of SYN and ACK should not be repeated. If the listen was not fully specified (i.e., the remote socket was not fully specified), then the unspecified fields should be filled in now.

fourth other text or control

Any other control or text-bearing segment (not containing SYN) must have an ACK and thus would be discarded by the ACK processing. An incoming RST segment could not be valid, since it could not have been sent in response to anything sent by this incarnation of the connection. So, if this unlikely condition is reached, the correct behavior is to drop the segment and return.

If the state is SYN-SENT then

first check the ACK bit

If the ACK bit is set

If SEG.ACK =< ISS, or SEG.ACK > SND.NXT, send a reset (unless the RST bit is set, if so drop the segment and return)

<SEQ=SEG.ACK><CTL=RST>

and discard the segment. Return.

If SND.UNA < SEG.ACK =< SND.NXT then the ACK is acceptable. Some deployed TCP code has used the check SEG.ACK == SND.NXT (using "==" rather than "=<", but this is not appropriate when the stack is capable of sending data on the SYN, because the TCP peer may not accept and acknowledge all of the data on the SYN.

second check the RST bit

If the RST bit is set

A potential blind reset attack is described in RFC 5961 [34]. The mitigation described in that document has specific applicability explained therein, and is not a substitute for cryptographic protection (e.g. IPsec or TCP-AO). A TCP implementation that supports the RFC 5961 mitigation SHOULD first check that the sequence number exactly matches RCV.NXT prior to executing the action in the next paragraph.

If the ACK was acceptable then signal the user "error: connection reset", drop the segment, enter CLOSED state, delete TCB, and return. Otherwise (no ACK) drop the segment and return.

third check the security

If the security/compartment in the segment does not exactly match the security/compartment in the TCB, send a reset

If there is an ACK

<SEQ=SEG.ACK><CTL=RST>

Otherwise

<SEQ=0><ACK=SEG.SEQ+SEG.LEN><CTL=RST,ACK>

If a reset was sent, discard the segment and return.

fourth check the SYN bit

This step should be reached only if the ACK is ok, or there is no ACK, and it the segment did not contain a RST.

If the SYN bit is on and the security/compartment is acceptable then, RCV.NXT is set to SEG.SEQ+1, IRS is set to SEG.SEQ. SND.UNA should be advanced to equal SEG.ACK (if there is an ACK), and any segments on the retransmission queue that are thereby acknowledged should be removed.

If SND.UNA > ISS (our SYN has been ACKed), change the connection state to ESTABLISHED, form an ACK segment

<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>

and send it. Data or controls that were queued for transmission may be included. If there are other controls or text in the segment then continue processing at the sixth step below where the URG bit is checked, otherwise return.

Otherwise enter SYN-RECEIVED, form a SYN,ACK segment

```
   <SEQ=ISS><ACK=RCV.NXT><CTL=SYN,ACK>
```

and send it. Set the variables:

```
   SND.WND <- SEG.WND
   SND.WL1 <- SEG.SEQ
   SND.WL2 <- SEG.ACK
```

If there are other controls or text in the segment, queue them for processing after the ESTABLISHED state has been reached, return.

Note that it is legal to send and receive application data on SYN segments (this is the "text in the segment" mentioned above. There has been significant misinformation and misunderstanding of this topic historically. Some firewalls and security devices consider this suspicious. However, the capability was used in T/TCP [17] and is used in TCP Fast Open (TFO) [44], so is important for implementations and network devices to permit.

fifth, if neither of the SYN or RST bits is set then drop the segment and return.

Otherwise,

first check sequence number

SYN-RECEIVED STATE
ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

Segments are processed in sequence. Initial tests on arrival are used to discard old duplicates, but further processing is done in SEG.SEQ order. If a segment's contents straddle the boundary between old and new, only the new parts should be processed.

In general, the processing of received segments MUST be implemented to aggregate ACK segments whenever possible (MUST-58). For example, if the TCP endpoint is processing a series of queued segments, it MUST process them all before sending any ACK segments (MUST-59).

There are four cases for the acceptability test for an incoming segment:

Segment Length	Receive Window	Test
0	0	SEG.SEQ = RCV.NXT
0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND
>0	0	not acceptable
>0	>0	RCV.NXT =< SEG.SEQ < RCV.NXT+RCV.WND or RCV.NXT =< SEG.SEQ+SEG.LEN-1 < RCV.NXT+RCV.WND

In implementing sequence number validation as described here, please note Appendix A.2.

If the RCV.WND is zero, no segments will be acceptable, but special allowance should be made to accept valid ACKs, URGs and RSTs.

If an incoming segment is not acceptable, an acknowledgment should be sent in reply (unless the RST bit is set, if so drop the segment and return):

```
<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>
```

After sending the acknowledgment, drop the unacceptable segment and return.

Note that for the TIME-WAIT state, there is an improved algorithm described in [36] for handling incoming SYN segments, that utilizes timestamps rather than relying on the sequence number check described here. When the improved algorithm is implemented, the logic above is not applicable for incoming SYN segments with timestamp options, received on a connection in the TIME-WAIT state.

In the following it is assumed that the segment is the idealized segment that begins at RCV.NXT and does not exceed the window. One could tailor actual segments to fit this assumption by trimming off any portions that lie outside the window (including SYN and FIN), and only processing further if the segment then begins at RCV.NXT. Segments with higher beginning sequence numbers SHOULD be held for later processing (SHLD-31).

second check the RST bit,

RFC 5961 [34] section 3 describes a potential blind reset attack and optional mitigation approach. This does not provide a cryptographic protection (e.g. as in IPsec or TCP-AO), but can be applicable in situations described in RFC 5961. For stacks implementing the RFC 5961 protection, the three checks below apply, otherwise processing for these states is indicated further below.

1) If the RST bit is set and the sequence number is outside the current receive window, silently drop the segment.

2) If the RST bit is set and the sequence number exactly matches the next expected sequence number (RCV.NXT), then TCP endpoints MUST reset the connection in the manner prescribed below according to the connection state.

3) If the RST bit is set and the sequence number does not exactly match the next expected sequence value, yet is within the current receive window, TCP endpoints MUST send an acknowledgement (challenge ACK):

```
<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>
```

After sending the challenge ACK, TCP endpoints MUST drop the unacceptable segment and stop processing the incoming packet further. Note that RFC 5961 and Errata ID 4772 contain additional considerations for ACK throttling in an implementation.

SYN-RECEIVED STATE

If the RST bit is set

If this connection was initiated with a passive OPEN (i.e., came from the LISTEN state), then return this connection to LISTEN state and return. The user need not be informed. If this connection was initiated with an active OPEN (i.e., came from SYN-SENT state) then the connection was refused, signal the user "connection refused". In either case, all segments on the retransmission queue should be removed. And in the active OPEN case, enter the CLOSED state and delete the TCB, and return.

ESTABLISHED
FIN-WAIT-1
FIN-WAIT-2
CLOSE-WAIT

If the RST bit is set then, any outstanding RECEIVEs and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

CLOSING STATE
LAST-ACK STATE
TIME-WAIT

If the RST bit is set then, enter the CLOSED state, delete the TCB, and return.

third check security

SYN-RECEIVED

If the security/compartment in the segment does not exactly match the security/compartment in the TCB then send a reset, and return.

ESTABLISHED
FIN-WAIT-1
FIN-WAIT-2
CLOSE-WAIT
CLOSING
LAST-ACK
TIME-WAIT

If the security/compartment in the segment does not exactly match the security/compartment in the TCB then send a reset, any outstanding RECEIVEs and SEND should receive "reset" responses. All segment queues should be flushed. Users should also receive an unsolicited general "connection reset" signal. Enter the CLOSED state, delete the TCB, and return.

Note this check is placed following the sequence check to prevent a segment from an old connection between these port numbers with a different security from causing an abort of the current connection.

fourth, check the SYN bit,

SYN-RECEIVED

If the connection was initiated with a passive OPEN, then return this connection to the LISTEN state and return. Otherwise, handle per the directions for synchronized states below.

ESTABLISHED STATE
FIN-WAIT STATE-1
FIN-WAIT STATE-2
CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

If the SYN bit is set in these synchronized states, it may be either a legitimate new connection attempt (e.g. in the case of TIME-WAIT), an error where the connection should be reset, or the result of an attack attempt, as described in RFC 5961 [34]. For the TIME-WAIT state, new connections can be accepted if the timestamp option is used and meets expectations (per [36]). For all other cases, RFC 5961 provides a mitigation with applicability to some situations, though there are also alternatives that offer cryptographic protection (see Section 6). RFC 5961 recommends that in these synchronized states, if the SYN bit is set, irrespective of the sequence number, TCP endpoints MUST send a "challenge ACK" to the remote peer:

```
<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>
```

After sending the acknowledgement, TCP implementations MUST drop the unacceptable segment and stop processing further. Note that RFC 5961 and Errata ID 4772 contain additional ACK throttling notes for an implementation.

For implementations that do not follow RFC 5961, the original RFC 793 behavior follows in this paragraph. If the SYN is in the window it is an error, send a reset, any outstanding RECEIVEs and SEND should receive "reset" responses, all segment queues should be flushed, the user should also receive an unsolicited general "connection reset" signal, enter the CLOSED state, delete the TCB, and return.

If the SYN is not in the window this step would not be reached and an ACK would have been sent in the first step (sequence number check).

fifth check the ACK field,

if the ACK bit is off drop the segment and return

if the ACK bit is on

RFC 5961 [34] section 5 describes a potential blind data injection attack, and mitigation that implementations MAY choose to include (MAY-12). TCP stacks that implement RFC 5961 MUST add an input check that the ACK value is acceptable only if it is in the range of ((SND.UNA - MAX.SND.WND) =< SEG.ACK =< SND.NXT). All incoming segments whose ACK value doesn't satisfy the above condition MUST be discarded and an ACK sent back. The new state variable MAX.SND.WND is defined as the largest window that the local sender has ever received from its peer (subject to window scaling) or may be hard-coded to a maximum permissible window value. When the ACK value is acceptable, the processing per-state below applies:

SYN-RECEIVED STATE

If SND.UNA < SEG.ACK =< SND.NXT then enter ESTABLISHED state and continue processing with variables below set to:

```
SND.WND <- SEG.WND
SND.WL1 <- SEG.SEQ
SND.WL2 <- SEG.ACK
```

If the segment acknowledgment is not acceptable, form a reset segment,

```
<SEQ=SEG.ACK><CTL=RST>
```

and send it.

ESTABLISHED STATE

If SND.UNA < SEG.ACK =< SND.NXT then, set SND.UNA <- SEG.ACK. Any segments on the retransmission queue that are thereby entirely acknowledged are removed. Users should receive positive acknowledgments for buffers that have been SENT and fully acknowledged (i.e., SEND buffer should be returned with "ok" response). If the ACK is a duplicate (SEG.ACK =< SND.UNA), it can be ignored. If the ACK acks something not yet sent (SEG.ACK > SND.NXT) then send an ACK, drop the segment, and return.

If SND.UNA =< SEG.ACK =< SND.NXT, the send window should be updated. If (SND.WL1 < SEG.SEQ or (SND.WL1 = SEG.SEQ and SND.WL2 =< SEG.ACK)), set SND.WND <- SEG.WND, set SND.WL1 <- SEG.SEQ, and set SND.WL2 <- SEG.ACK.

Note that SND.WND is an offset from SND.UNA, that SND.WL1 records the sequence number of the last segment used to update SND.WND, and that SND.WL2 records the acknowledgment number of the last segment used to update SND.WND. The check here prevents using old segments to update the window.

FIN-WAIT-1 STATE

In addition to the processing for the ESTABLISHED state, if the FIN segment is now acknowledged then enter FIN-WAIT-2 and continue processing in that state.

FIN-WAIT-2 STATE

In addition to the processing for the ESTABLISHED state, if the retransmission queue is empty, the user’s CLOSE can be acknowledged ("ok") but do not delete the TCB.

CLOSE-WAIT STATE

Do the same processing as for the ESTABLISHED state.

CLOSING STATE

In addition to the processing for the ESTABLISHED state, if the ACK acknowledges our FIN then enter the TIME-WAIT state, otherwise ignore the segment.

LAST-ACK STATE

The only thing that can arrive in this state is an acknowledgment of our FIN. If our FIN is now acknowledged, delete the TCB, enter the CLOSED state, and return.

TIME-WAIT STATE

The only thing that can arrive in this state is a retransmission of the remote FIN. Acknowledge it, and restart the 2 MSL timeout.

sixth, check the URG bit,

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

If the URG bit is set, RCV.UP <- max(RCV.UP,SEG.UP), and signal the user that the remote side has urgent data if the urgent pointer (RCV.UP) is in advance of the data consumed. If the user has already been signaled (or is still in the "urgent mode") for this continuous sequence of urgent data, do not signal the user again.

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT

This should not occur, since a FIN has been received from the remote side. Ignore the URG.

seventh, process the segment text,

ESTABLISHED STATE
FIN-WAIT-1 STATE
FIN-WAIT-2 STATE

Once in the ESTABLISHED state, it is possible to deliver segment text to user RECEIVE buffers. Text from segments can be moved into buffers until either the buffer is full or the segment is empty. If the segment empties and carries a PUSH flag, then the user is informed, when the buffer is returned, that a PUSH has been received.

When the TCP endpoint takes responsibility for delivering the data to the user it must also acknowledge the receipt of the data.

Once the TCP endpoint takes responsibility for the data it advances RCV.NXT over the data accepted, and adjusts RCV.WND as appropriate to the current buffer availability. The total of RCV.NXT and RCV.WND should not be reduced.

A TCP implementation MAY send an ACK segment acknowledging RCV.NXT when a valid segment arrives that is in the window but not at the left window edge (MAY-13).

Please note the window management suggestions in Section 3.7.

Send an acknowledgment of the form:

```
<SEQ=SND.NXT><ACK=RCV.NXT><CTL=ACK>
```

This acknowledgment should be piggybacked on a segment being transmitted if possible without incurring undue delay.

CLOSE-WAIT STATE
CLOSING STATE
LAST-ACK STATE
TIME-WAIT STATE

This should not occur, since a FIN has been received from the remote side. Ignore the segment text.

eighth, check the FIN bit,

Do not process the FIN if the state is CLOSED, LISTEN or SYN-SENT since the SEG.SEQ cannot be validated; drop the segment and return.

If the FIN bit is set, signal the user "connection closing" and return any pending RECEIVEs with same message, advance RCV.NXT over the FIN, and send an acknowledgment for the FIN. Note that FIN implies PUSH for any segment text not yet delivered to the user.

SYN-RECEIVED STATE
ESTABLISHED STATE

Enter the CLOSE-WAIT state.

FIN-WAIT-1 STATE

If our FIN has been ACKed (perhaps in this segment), then enter TIME-WAIT, start the time-wait timer, turn off the other timers; otherwise enter the CLOSING state.

FIN-WAIT-2 STATE

Enter the TIME-WAIT state. Start the time-wait timer, turn off the other timers.

CLOSE-WAIT STATE

Remain in the CLOSE-WAIT state.

CLOSING STATE

Remain in the CLOSING state.

LAST-ACK STATE

Remain in the LAST-ACK state.

TIME-WAIT STATE

Remain in the TIME-WAIT state. Restart the 2 MSL time-wait timeout.

and return.

USER TIMEOUT

USER TIMEOUT

For any state if the user timeout expires, flush all queues, signal the user "error: connection aborted due to user timeout" in general and for any outstanding calls, delete the TCB, enter the CLOSED state and return.

RETRANSMISSION TIMEOUT

For any state if the retransmission timeout expires on a segment in the retransmission queue, send the segment at the front of the retransmission queue again, reinitialize the retransmission timer, and return.

TIME-WAIT TIMEOUT

If the time-wait timeout expires on a connection delete the TCB, enter the CLOSED state and return.

3.10. Glossary

ACK
A control bit (acknowledge) occupying no sequence space, which indicates that the acknowledgment field of this segment specifies the next sequence number the sender of this segment is expecting to receive, hence acknowledging receipt of all previous sequence numbers.

connection
A logical communication path identified by a pair of sockets.

datagram
A message sent in a packet switched computer communications network.

Destination Address
The network layer address of the remote endpoint.

FIN
A control bit (finis) occupying one sequence number, which indicates that the sender will send no more data or control occupying sequence space.

fragment
A portion of a logical unit of data, in particular an internet fragment is a portion of an internet datagram.

header
Control information at the beginning of a message, segment, fragment, packet or block of data.

host
A computer. In particular a source or destination of messages from the point of view of the communication network.

Identification
An Internet Protocol field. This identifying value assigned by the sender aids in assembling the fragments of a datagram.

internet address
A network layer address.

internet datagram
The unit of data exchanged between an internet module and the higher level protocol together with the internet header.

internet fragment

A portion of the data of an internet datagram with an internet header.

IP

Internet Protocol. See [1] and [12].

IRS

The Initial Receive Sequence number. The first sequence number used by the sender on a connection.

ISN

The Initial Sequence Number. The first sequence number used on a connection, (either ISS or IRS). Selected in a way that is unique within a given period of time and is unpredictable to attackers.

ISS

The Initial Send Sequence number. The first sequence number used by the sender on a connection.

left sequence

This is the next sequence number to be acknowledged by the data receiving TCP endpoint (or the lowest currently unacknowledged sequence number) and is sometimes referred to as the left edge of the send window.

module

An implementation, usually in software, of a protocol or other procedure.

MSL

Maximum Segment Lifetime, the time a TCP segment can exist in the internetwork system. Arbitrarily defined to be 2 minutes.

octet

An eight bit byte.

Options

An Option field may contain several options, and each option may be several octets in length.

packet

A package of data with a header that may or may not be logically complete. More often a physical packaging than a logical packaging of data.

port

The portion of a connection identifier used for demultiplexing connections at an endpoint.

process
A program in execution. A source or destination of data from the point of view of the TCP endpoint or other host-to-host protocol.

PUSH
A control bit occupying no sequence space, indicating that this segment contains data that must be pushed through to the receiving user.

RCV.NXT
receive next sequence number

RCV.UP
receive urgent pointer

RCV.WND
receive window

receive next sequence number
This is the next sequence number the local TCP endpoint is expecting to receive.

receive window
This represents the sequence numbers the local (receiving) TCP endpoint is willing to receive. Thus, the local TCP endpoint considers that segments overlapping the range RCV.NXT to RCV.NXT + RCV.WND - 1 carry acceptable data or control. Segments containing sequence numbers entirely outside of this range are considered duplicates and discarded.

RST
A control bit (reset), occupying no sequence space, indicating that the receiver should delete the connection without further interaction. The receiver can determine, based on the sequence number and acknowledgment fields of the incoming segment, whether it should honor the reset command or ignore it. In no case does receipt of a segment containing RST give rise to a RST in response.

SEG.ACK
segment acknowledgment

SEG.LEN

segment length

SEG.SEQ
segment sequence

SEG.UP
segment urgent pointer field

SEG.WND
segment window field

segment
A logical unit of data, in particular a TCP segment is the unit of data transferred between a pair of TCP modules.

segment acknowledgment
The sequence number in the acknowledgment field of the arriving segment.

segment length
The amount of sequence number space occupied by a segment, including any controls that occupy sequence space.

segment sequence
The number in the sequence field of the arriving segment.

send sequence
This is the next sequence number the local (sending) TCP endpoint will use on the connection. It is initially selected from an initial sequence number curve (ISN) and is incremented for each octet of data or sequenced control transmitted.

send window
This represents the sequence numbers that the remote (receiving) TCP endpoint is willing to receive. It is the value of the window field specified in segments from the remote (data receiving) TCP endpoint. The range of new sequence numbers that may be emitted by a TCP implementation lies between SND.NXT and SND.UNA + SND.WND - 1. (Retransmissions of sequence numbers between SND.UNA and SND.NXT are expected, of course.)

SND.NXT
send sequence

SND.UNA
left sequence

SND.UP
        send urgent pointer

SND.WL1
        segment sequence number at last window update

SND.WL2
        segment acknowledgment number at last window update

SND.WND
        send window

socket (or socket number, or socket address, or socket identifier)
        An address that specifically includes a port identifier, that is, the concatenation of an Internet Address with a TCP port.

Source Address
        The network layer address of the sending endpoint.

SYN
        A control bit in the incoming segment, occupying one sequence number, used at the initiation of a connection, to indicate where the sequence numbering will start.

TCB
        Transmission control block, the data structure that records the state of a connection.

TCP
        Transmission Control Protocol: A host-to-host protocol for reliable communication in internetwork environments.

TOS
        Type of Service, an obsoleted IPv4 field. The same header bits currently are used for the Differentiated Services field [5] containing the Differentiated Services Code Point (DSCP) value and the 2-bit ECN codepoint [8].

Type of Service
        An Internet Protocol field that indicates the type of service for this internet fragment.

URG
        A control bit (urgent), occupying no sequence space, used to indicate that the receiving user should be notified to do urgent processing as long as there is data to be consumed with sequence numbers less than the value indicated in the urgent pointer.

urgent pointer
A control field meaningful only when the URG bit is on. This field communicates the value of the urgent pointer that indicates the data octet associated with the sending user's urgent call.

4. Changes from RFC 793

This document obsoletes RFC 793 as well as RFC 6093 and 6528, which updated 793. In all cases, only the normative protocol specification and requirements have been incorporated into this document, and some informational text with background and rationale may not have been carried in. The informational content of those documents is still valuable in learning about and understanding TCP, and they are valid Informational references, even though their normative content has been incorporated into this document.

The main body of this document was adapted from RFC 793's Section 3, titled "FUNCTIONAL SPECIFICATION", with an attempt to keep formatting and layout as close as possible.

The collection of applicable RFC Errata that have been reported and either accepted or held for an update to RFC 793 were incorporated (Errata IDs: 573, 574, 700, 701, 1283, 1561, 1562, 1564, 1565, 1571, 1572, 2296, 2297, 2298, 2748, 2749, 2934, 3213, 3300, 3301, 6222). Some errata were not applicable due to other changes (Errata IDs: 572, 575, 1569, 3305, 3602).

Changes to the specification of the Urgent Pointer described in RFC 1122 and 6093 were incorporated. See RFC 6093 for detailed discussion of why these changes were necessary.

The discussion of the RTO from RFC 793 was updated to refer to RFC 6298. The RFC 1122 text on the RTO originally replaced the 793 text, however, RFC 2988 should have updated 1122, and has subsequently been obsoleted by 6298.

RFC 1122 contains a collection of other changes and clarifications to RFC 793. The normative items impacting the protocol have been incorporated here, though some historically useful implementation advice and informative discussion from RFC 1122 is not included here.

RFC 1122 contains more than just TCP requirements, so this document can't obsolete RFC 1122 entirely. It is only marked as "updating" 1122, however, it should be understood to effectively obsolete all of the RFC 1122 material on TCP.

The more secure Initial Sequence Number generation algorithm from RFC 6528 was incorporated. See RFC 6528 for discussion of the attacks that this mitigates, as well as advice on selecting PRF algorithms and managing secret key data.

A note based on RFC 6429 was added to explicitly clarify that system resource management concerns allow connection resources to be reclaimed. RFC 6429 is obsoleted in the sense that this clarification has been reflected in this update to the base TCP specification now.

RFC EDITOR'S NOTE: the content below is for detailed change tracking and planning, and not to be included with the final revision of the document.

This document started as draft-eddy-rfc793bis-00, that was merely a proposal and rough plan for updating RFC 793.

The -01 revision of this draft-eddy-rfc793bis incorporates the content of RFC 793 Section 3 titled "FUNCTIONAL SPECIFICATION". Other content from RFC 793 has not been incorporated. The -01 revision of this document makes some minor formatting changes to the RFC 793 content in order to convert the content into XML2RFC format and account for left-out parts of RFC 793. For instance, figure numbering differs and some indentation is not exactly the same.

The -02 revision of draft-eddy-rfc793bis incorporates errata that have been verified:

Errata ID 573: Reported by Bob Braden (note: This errata basically is just a reminder that RFC 1122 updates 793. Some of the associated changes are left pending to a separate revision that incorporates 1122. Bob's mention of PUSH in 793 section 2.8 was not applicable here because that section was not part of the "functional specification". Also the 1122 text on the retransmission timeout also has been updated by subsequent RFCs, so the change here deviates from Bob's suggestion to apply the 1122 text.)
Errata ID 574: Reported by Yin Shuming
Errata ID 700: Reported by Yin Shuming
Errata ID 701: Reported by Yin Shuming
Errata ID 1283: Reported by Pei-chun Cheng
Errata ID 1561: Reported by Constantin Hagemeier
Errata ID 1562: Reported by Constantin Hagemeier
Errata ID 1564: Reported by Constantin Hagemeier
Errata ID 1565: Reported by Constantin Hagemeier
Errata ID 1571: Reported by Constantin Hagemeier
Errata ID 1572: Reported by Constantin Hagemeier

Errata ID 2296: Reported by Vishwas Manral
Errata ID 2297: Reported by Vishwas Manral
Errata ID 2298: Reported by Vishwas Manral
Errata ID 2748: Reported by Mykyta Yevstifeyev
Errata ID 2749: Reported by Mykyta Yevstifeyev
Errata ID 2934: Reported by Constantin Hagemeier
Errata ID 3213: Reported by EugnJun Yi
Errata ID 3300: Reported by Botong Huang
Errata ID 3301: Reported by Botong Huang
Errata ID 3305: Reported by Botong Huang
Note: Some verified errata were not used in this update, as they relate to sections of RFC 793 elided from this document. These include Errata ID 572, 575, and 1569.
Note: Errata ID 3602 was not applied in this revision as it is duplicative of the 1122 corrections.

Not related to RFC 793 content, this revision also makes small tweaks to the introductory text, fixes indentation of the pseudo header diagram, and notes that the Security Considerations should also include privacy, when this section is written.

The -03 revision of draft-eddy-rfc793bis revises all discussion of the urgent pointer in order to comply with RFC 6093, 1122, and 1011. Since 1122 held requirements on the urgent pointer, the full list of requirements was brought into an appendix of this document, so that it can be updated as-needed.

The -04 revision of draft-eddy-rfc793bis includes the ISN generation changes from RFC 6528.

The -05 revision of draft-eddy-rfc793bis incorporates MSS requirements and definitions from RFC 879, 1122, and 6691, as well as option-handling requirements from RFC 1122.

The -00 revision of draft-ietf-tcpm-rfc793bis incorporates several additional clarifications and updates to the section on segmentation, many of which are based on feedback from Joe Touch improving from the initial text on this in the previous revision.

The -01 revision incorporates the change to Reserved bits due to ECN, as well as many other changes that come from RFC 1122.

The -02 revision has small formatting modifications in order to address xml2rfc warnings about long lines. It was a quick update to avoid document expiration. TCPM working group discussion in 2015 also indicated that that we should not try to add sections on implementation advice or similar non-normative information.

The -03 revision incorporates more content from RFC 1122: Passive OPEN Calls, Time-To-Live, Multihoming, IP Options, ICMP messages, Data Communications, When to Send Data, When to Send a Window Update, Managing the Window, Probing Zero Windows, When to Send an ACK Segment. The section on data communications was re-organized into clearer subsections (previously headings were embedded in the 793 text), and windows management advice from 793 was removed (as reviewed by TCPM working group) in favor of the 1122 additions on SWS, ZWP, and related topics.

The -04 revision includes reference to RFC 6429 on the ZWP condition, RFC1122 material on TCP Connection Failures, TCP Keep-Alives, Acknowledging Queued Segments, and Remote Address Validation. RTO computation is referenced from RFC 6298 rather than RFC 1122.

The -05 revision includes the requirement to implement TCP congestion control with recommendation to implement ECN, the RFC 6633 update to 1122, which changed the requirement on responding to source quench ICMP messages, and discussion of ICMP (and ICMPv6) soft and hard errors per RFC 5461 (ICMPv6 handling for TCP doesn’t seem to be mentioned elsewhere in standards track).

The -06 revision includes an appendix on "Other Implementation Notes" to capture widely-deployed fundamental features that are not contained in the RFC series yet. It also added mention of RFC 6994 and the IANA TCP parameters registry as a reference. It includes references to RFC 5961 in appropriate places. The references to TOS were changed to DiffServ field, based on reflecting RFC 2474 as well as the IPv6 presence of traffic class (carrying DiffServ field) rather than TOS.

The -07 revision includes reference to RFC 6191, updated security considerations, discussion of additional implementation considerations, and clarification of data on the SYN.

The -08 revision includes changes based on:

describing treatment of reserved bits (following TCPM mailing list thread from July 2014 on "793bis item - reserved bit behavior"
addition a brief TCP key concepts section to make up for not including the outdated section 2 of RFC 793
changed "TCP" to "host" to resolve conflict between 1122 wording on whether TCP or the network layer chooses an address when multihomed
fixed/updated definition of options in glossary
moved note on aggregating ACKs from 1122 to a more appropriate location
resolved notes on IP precedence and security/compartment

added implementation note on sequence number validation

added note that PUSH does not apply when Nagle is active

added 1122 content on asynchronous reports to replace 793 section on TCP to user messages

The -09 revision fixes section numbering problems.

The -10 revision includes additions to the security considerations based on comments from Joe Touch, and suggested edits on RST/FIN notification, RFC 2525 reference, and other edits suggested by Yuchung Cheng, as well as modifications to DiffServ text from Yuchung Cheng and Gorry Fairhurst.

The -11 revision includes a start at identifying all of the requirements text and referencing each instance in the common table at the end of the document.

The -12 revision completes the requirement language indexing started in -11 and adds necessary description of the PUSH functionality that was missing.

The -13 revision contains only changes in the inline editor notes.

The -14 revision includes updates with regard to several comments from the mailing list, including editorial fixes, adding IANA considerations for the header flags, improving figure title placement, and breaking up the "Terminology" section into more appropriately titled subsections.

The -15 revision has many technical and editorial corrections from Gorry Fairhurst's review, and subsequent discussion on the TCPM list, as well as some other collected clarifications and improvements from mailing list discussion.

The -16 revision addresses several discussions that rose from additional reviews and follow-up on some of Gorry Fairhurst's comments from revision 14.

The -17 revision includes errata 6222 from Charles Deng, update to the key words boilerplate, updated description of the header flags registry changes, and clarification about connections rather than users in the discussion of OPEN calls.

The -18 revision includes editorial changes to the IANA considerations, based on comments from Richard Scheffenegger at the IETF 108 TCPM virtual meeting.

The -19 revision includes editorial changes from Errata 6281 and 6282 reported by Merlin Buge. It also includes WGLC changes noted by Mohamed Boucadair, Rahul Jadhav, Praveen Balasubramanian, Matt Olson, Yi Huang, Joe Touch, and Juhamatti Kuusisaari.

Some other suggested changes that will not be incorporated in this 793 update unless TCPM consensus changes with regard to scope are:

1. Tony Sabatini's suggestion for describing DO field
2. Per discussion with Joe Touch (TAPS list, 6/20/2015), the description of the API could be revisited

Early in the process of updating RFC 793, Scott Brim mentioned that this should include a PERPASS/privacy review. This may be something for the chairs or AD to request during WGLC or IETF LC.

5. IANA Considerations

In the "Transmission Control Protocol (TCP) Header Flags" registry, IANA is asked to make several changes described in this section.

RFC 3168 originally created this registry, but only populated it with the new bits defined in RFC 3168, neglecting the other bits that had previously been described in RFC 793 and other documents. Bit 7 has since also been updated by RFC 8311.

The "Bit" column is renamed below as the "Bit Offset" column, since it references each header flag's offset within the 16-bit aligned view of the TCP header in Figure 1. The bits in offsets 0 through 4 are the TCP segment Data Offset field, and not header flags.

IANA should add a column for "Assignment Notes".

IANA should assign values indicated below.

TCP Header Flags

Bit Offset	Name	Reference	Assignment Notes
4	Reserved for future use	(this document)	
5	Reserved for future use	(this document)	
6	Reserved for future use	(this document)	
7	Reserved for future use	[RFC8311]	Previously used by Historic [RFC3540] as NS (Nonce Sum)
8	CWR (Congestion Window Reduced)	[RFC3168]	
9	ECE (ECN-Echo)	[RFC3168]	
10	Urgent Pointer field significant (URG)	(this document)	
11	Acknowledgment field significant (ACK)	(this document)	
12	Push Function (PSH)	(this document)	
13	Reset the connection (RST)	(this document)	
14	Synchronize sequence numbers (SYN)	(this document)	
15	No more data from sender (FIN)	(this document)	

This TCP Header Flags registry should also be moved to a sub-registry under the global "Transmission Control Protocol (TCP) Parameters registry (https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml).

The registry's Registration Procedure should remain Standards Action, but the Reference can be updated to this document, and the Note removed.

6. Security and Privacy Considerations

The TCP design includes only rudimentary security features that improve the robustness and reliability of connections and application data transfer, but there are no built-in cryptographic capabilities to support any form of privacy, authentication, or other typical security functions. Non-cryptographic enhancements (e.g. [34]) have been developed to improve robustness of TCP connections to particular types of attacks, but the applicability and protections of non-cryptographic enhancements are limited (e.g. see section 1.1 of [34]). Applications typically utilize lower-layer (e.g. IPsec) and upper-layer (e.g. TLS) protocols to provide security and privacy for TCP connections and application data carried in TCP. Methods based on TCP options have been developed as well, to support some security capabilities.

In order to fully protect TCP connections (including their control flags) IPsec or the TCP Authentication Option (TCP-AO) [33] are the only current effective methods. Other methods discussed in this section may protect the payload, but either only a subset of the fields (e.g. tcpcrypt [56]) or none at all (e.g. TLS). Other security features that have been added to TCP (e.g. ISN generation, sequence number checks, and others) are only capable of partially hindering attacks.

Applications using long-lived TCP flows have been vulnerable to attacks that exploit the processing of control flags described in earlier TCP specifications [27]. TCP-MD5 was a commonly implemented TCP option to support authentication for some of these connections, but had flaws and is now deprecated. TCP-AO provides a capability to protect long-lived TCP connections from attacks, and has superior properties to TCP-MD5. It does not provide any privacy for application data, nor for the TCP headers.

The "tcpcrypt" [56] Experimental extension to TCP provides the ability to cryptographically protect connection data. Metadata aspects of the TCP flow are still visible, but the application stream is well-protected. Within the TCP header, only the urgent pointer and FIN flag are protected through tcpcrypt.

The TCP Roadmap [45] includes notes about several RFCs related to TCP security. Many of the enhancements provided by these RFCs have been integrated into the present document, including ISN generation, mitigating blind in-window attacks, and improving handling of soft errors and ICMP packets. These are all discussed in greater detail in the referenced RFCs that originally described the changes needed to earlier TCP specifications. Additionally, see RFC 6093 [35] for discussion of security considerations related to the urgent pointer field, that has been deprecated.

Since TCP is often used for bulk transfer flows, some attacks are possible that abuse the TCP congestion control logic. An example is "ACK-division" attacks. Updates that have been made to the TCP congestion control specifications include mechanisms like Appropriate Byte Counting (ABC) [23] that act as mitigations to these attacks.

Other attacks are focused on exhausting the resources of a TCP server. Examples include SYN flooding [26] or wasting resources on non-progressing connections [37]. Operating systems commonly implement mitigations for these attacks. Some common defenses also utilize proxies, stateful firewalls, and other technologies outside of the end-host TCP implementation.

7. Acknowledgements

This document is largely a revision of RFC 793, which Jon Postel was the editor of. Due to his excellent work, it was able to last for three decades before we felt the need to revise it.

Andre Oppermann was a contributor and helped to edit the first revision of this document.

We are thankful for the assistance of the IETF TCPM working group chairs, over the course of work on this document:

Michael Scharf
Yoshifumi Nishida
Pasi Sarolahti
Michael Tuexen

During the discussions of this work on the TCPM mailing list and in working group meetings, helpful comments, critiques, and reviews were received from (listed alphabetically by last name): Praveen Balasubramanian, David Borman, Mohamed Boucadair, Bob Briscoe, Neal Cardwell, Yuchung Cheng, Martin Duke, Ted Faber, Gorry Fairhurst, Fernando Gont, Rodney Grimes, Yi Huang, Rahul Jadhav, Mike Kosek, Juhamatti Kuusisaari, Kevin Lahey, Kevin Mason, Matt Mathis, Jonathan Morton, Matt Olson, Tommy Pauly, Tom Petch, Hagen Paul Pfeifer, Anthony Sabatini, Michael Scharf, Greg Skinner, Joe Touch, Michael Tuexen, Reji Varghese, Tim Wicinski, Lloyd Wood, and Alex Zimmermann. Joe Touch provided additional help in clarifying the description of segment size parameters and PMTUD/PLPMTUD recommendations.

This document includes content from errata that were reported by (listed chronologically): Yin Shuming, Bob Braden, Morris M. Keesan, Pei-chun Cheng, Constantin Hagemeier, Vishwas Manral, Mykyta Yevstifeyev, EungJun Yi, Botong Huang, Charles Deng, Merlin Buge.

8. References

8.1. Normative References

[1] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/info/rfc791>.

[2] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, DOI 10.17487/RFC1191, November 1990, <https://www.rfc-editor.org/info/rfc1191>.

[3] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, DOI 10.17487/RFC1981, August 1996, <https://www.rfc-editor.org/info/rfc1981>.

[4] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[5] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, DOI 10.17487/RFC2474, December 1998, <https://www.rfc-editor.org/info/rfc2474>.

[6] Borman, D., Deering, S., and R. Hinden, "IPv6 Jumbograms", RFC 2675, DOI 10.17487/RFC2675, August 1999, <https://www.rfc-editor.org/info/rfc2675>.

[7] Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, DOI 10.17487/RFC2923, September 2000, <https://www.rfc-editor.org/info/rfc2923>.

[8] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, DOI 10.17487/RFC3168, September 2001, <https://www.rfc-editor.org/info/rfc3168>.

[9] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP’s Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[10] Gont, F., "Deprecation of ICMP Source Quench Messages", RFC 6633, DOI 10.17487/RFC6633, May 2012, <https://www.rfc-editor.org/info/rfc6633>.

[11] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[12] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>.

8.2. Informative References

[13] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[14] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <https://www.rfc-editor.org/info/rfc896>.

[15] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <https://www.rfc-editor.org/info/rfc1122>.

[16] Almquist, P., "Type of Service in the Internet Protocol Suite", RFC 1349, DOI 10.17487/RFC1349, July 1992, <https://www.rfc-editor.org/info/rfc1349>.

[17] Braden, R., "T/TCP -- TCP Extensions for Transactions Functional Specification", RFC 1644, DOI 10.17487/RFC1644, July 1994, <https://www.rfc-editor.org/info/rfc1644>.

[18] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <https://www.rfc-editor.org/info/rfc2018>.

[19] Paxson, V., Allman, M., Dawson, S., Fenner, W., Griner, J., Heavens, I., Lahey, K., Semke, J., and B. Volz, "Known TCP Implementation Problems", RFC 2525, DOI 10.17487/RFC2525, March 1999, <https://www.rfc-editor.org/info/rfc2525>.

[20] Xiao, X., Hannan, A., Paxson, V., and E. Crabbe, "TCP Processing of the IPv4 Precedence Field", RFC 2873, DOI 10.17487/RFC2873, June 2000, <https://www.rfc-editor.org/info/rfc2873>.

[21] Floyd, S., Mahdavi, J., Mathis, M., and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP", RFC 2883, DOI 10.17487/RFC2883, July 2000, <https://www.rfc-editor.org/info/rfc2883>.

[22] Balakrishnan, H., Padmanabhan, V., Fairhurst, G., and M. Sooriyabandara, "TCP Performance Implications of Network Path Asymmetry", BCP 69, RFC 3449, DOI 10.17487/RFC3449, December 2002, <https://www.rfc-editor.org/info/rfc3449>.

[23] Allman, M., "TCP Congestion Control with Appropriate Byte Counting (ABC)", RFC 3465, DOI 10.17487/RFC3465, February 2003, <https://www.rfc-editor.org/info/rfc3465>.

[24] Fenner, B., "Experimental Values In IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers", RFC 4727, DOI 10.17487/RFC4727, November 2006, <https://www.rfc-editor.org/info/rfc4727>.

[25] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <https://www.rfc-editor.org/info/rfc4821>.

[26] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <https://www.rfc-editor.org/info/rfc4987>.

[27] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, DOI 10.17487/RFC4953, July 2007, <https://www.rfc-editor.org/info/rfc4953>.

[28] Culley, P., Elzur, U., Recio, R., Bailey, S., and J. Carrier, "Marker PDU Aligned Framing for TCP Specification", RFC 5044, DOI 10.17487/RFC5044, October 2007, <https://www.rfc-editor.org/info/rfc5044>.

[29] Gont, F., "TCP's Reaction to Soft Errors", RFC 5461, DOI 10.17487/RFC5461, February 2009, <https://www.rfc-editor.org/info/rfc5461>.

[30] StJohns, M., Atkinson, R., and G. Thomas, "Common Architecture Label IPv6 Security Option (CALIPSO)", RFC 5570, DOI 10.17487/RFC5570, July 2009, <https://www.rfc-editor.org/info/rfc5570>.

[31] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[32] Sandlund, K., Pelletier, G., and L-E. Jonsson, "The RObust Header Compression (ROHC) Framework", RFC 5795, DOI 10.17487/RFC5795, March 2010, <https://www.rfc-editor.org/info/rfc5795>.

[33] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <https://www.rfc-editor.org/info/rfc5925>.

[34] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <https://www.rfc-editor.org/info/rfc5961>.

[35] Gont, F. and A. Yourtchenko, "On the Implementation of the TCP Urgent Mechanism", RFC 6093, DOI 10.17487/RFC6093, January 2011, <https://www.rfc-editor.org/info/rfc6093>.

[36] Gont, F., "Reducing the TIME-WAIT State Using TCP Timestamps", BCP 159, RFC 6191, DOI 10.17487/RFC6191, April 2011, <https://www.rfc-editor.org/info/rfc6191>.

[37] Bashyam, M., Jethanandani, M., and A. Ramaiah, "TCP Sender Clarification for Persist Condition", RFC 6429, DOI 10.17487/RFC6429, December 2011, <https://www.rfc-editor.org/info/rfc6429>.

[38] Gont, F. and S. Bellovin, "Defending against Sequence Number Attacks", RFC 6528, DOI 10.17487/RFC6528, February 2012, <https://www.rfc-editor.org/info/rfc6528>.

[39] Borman, D., "TCP Options and Maximum Segment Size (MSS)", RFC 6691, DOI 10.17487/RFC6691, July 2012, <https://www.rfc-editor.org/info/rfc6691>.

[40] Touch, J., "Updated Specification of the IPv4 ID Field", RFC 6864, DOI 10.17487/RFC6864, February 2013, <https://www.rfc-editor.org/info/rfc6864>.

[41] Touch, J., "Shared Use of Experimental TCP Options", RFC 6994, DOI 10.17487/RFC6994, August 2013, <https://www.rfc-editor.org/info/rfc6994>.

[42] McPherson, D., Oran, D., Thaler, D., and E. Osterweil, "Architectural Considerations of IP Anycast", RFC 7094, DOI 10.17487/RFC7094, January 2014, <https://www.rfc-editor.org/info/rfc7094>.

[43] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <https://www.rfc-editor.org/info/rfc7323>.

[44] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.

[45] Duke, M., Braden, R., Eddy, W., Blanton, E., and A. Zimmermann, "A Roadmap for Transmission Control Protocol (TCP) Specification Documents", RFC 7414, DOI 10.17487/RFC7414, February 2015, <https://www.rfc-editor.org/info/rfc7414>.

[46] Black, D., Ed. and P. Jones, "Differentiated Services (Diffserv) and Real-Time Communication", RFC 7657, DOI 10.17487/RFC7657, November 2015, <https://www.rfc-editor.org/info/rfc7657>.

[47] Fairhurst, G. and M. Welzl, "The Benefits of Using Explicit Congestion Notification (ECN)", RFC 8087, DOI 10.17487/RFC8087, March 2017, <https://www.rfc-editor.org/info/rfc8087>.

[48] Fairhurst, G., Ed., Trammell, B., Ed., and M. Kuehlewind, Ed., "Services Provided by IETF Transport Protocols and Congestion Control Mechanisms", RFC 8095, DOI 10.17487/RFC8095, March 2017, <https://www.rfc-editor.org/info/rfc8095>.

[49] Welzl, M., Tuexen, M., and N. Khademi, "On the Usage of Transport Features Provided by IETF Transport Protocols", RFC 8303, DOI 10.17487/RFC8303, February 2018, <https://www.rfc-editor.org/info/rfc8303>.

[50] Chown, T., Loughney, J., and T. Winters, "IPv6 Node Requirements", BCP 220, RFC 8504, DOI 10.17487/RFC8504, January 2019, <https://www.rfc-editor.org/info/rfc8504>.

[51] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <https://www.rfc-editor.org/info/rfc8684>.

[52] IANA, "Transmission Control Protocol (TCP) Parameters, https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xhtml", 2019.

[53] IANA, "Transmission Control Protocol (TCP) Header Flags, https://www.iana.org/assignments/tcp-header-flags/tcp-header-flags.xhtml", 2019.

[54] Gont, F., "Processing of IP Security/Compartment and Precedence Information by TCP", draft-gont-tcpm-tcp-seccomp-prec-00 (work in progress), March 2012.

[55] Gont, F. and D. Borman, "On the Validation of TCP Sequence Numbers", draft-gont-tcpm-tcp-seq-validation-02 (work in progress), March 2015.

[56] Bittau, A., Giffin, D., Handley, M., Mazieres, D., Slack, Q., and E. Smith, "Cryptographic protection of TCP Streams (tcpcrypt)", draft-ietf-tcpinc-tcpcrypt-09 (work in progress), November 2017.

[57] Touch, J. and W. Eddy, "TCP Extended Data Offset Option", draft-ietf-tcpm-tcp-edo-10 (work in progress), July 2018.

[58] Minshall, G., "A Proposed Modification to Nagle's Algorithm", draft-minshall-nagle-01 (work in progress), June 1999.

[59] Dalal, Y. and C. Sunshine, "Connection Management in Transport Protocols", Computer Networks Vol. 2, No. 6, pp. 454-473, December 1978.

[60] Faber, T., Touch, J., and W. Yui, "The TIME-WAIT state in TCP and Its Effect on Busy Servers", Proceedings of IEEE INFOCOM pp. 1573-1583, March 1999.

Appendix A. Other Implementation Notes

This section includes additional notes and references on TCP implementation decisions that are currently not a part of the RFC series or included within the TCP standard. These items can be considered by implementers, but there was not yet a consensus to include them in the standard.

A.1. IP Security Compartment and Precedence

The IPv4 specification [1] includes a precedence value in the (now obsoleted) Type of Service field (TOS) field. It was modified in [16], and then obsoleted by the definition of Differentiated Services (DiffServ) [5]. Setting and conveying TOS between the network layer, TCP implementation, and applications is obsolete, and replaced by DiffServ in the current TCP specification.

RFC 793 requires checking the IP security compartment and precedence on incoming TCP segments for consistency within a connection, and with application requests. Each of these aspects of IP have become outdated, without specific updates to RFC 793. The issues with precedence were fixed by [20], which is Standards Track, and so this present TCP specification includes those changes. However, the state of IP security options that may be used by MLS systems is not as clean.

Resetting connections when incoming packets do not meet expected security compartment or precedence expectations has been recognized as a possible attack vector [54], and there has been discussion about amending the TCP specification to prevent connections from being aborted due to non-matching IP security compartment and DiffServ codepoint values.

A.1.1. Precedence

In DiffServ the former precedence values are treated as Class Selector codepoints, and methods for compatible treatment are described in the DiffServ architecture. The RFC 793/1122 TCP specification includes logic intending to have connections use the highest precedence requested by either endpoint application, and to keep the precedence consistent throughout a connection. This logic from the obsolete TOS is not applicable for DiffServ, and should not be included in TCP implementations, though changes to DiffServ values within a connection are discouraged. For discussion of this, see RFC 7657 (sec 5.1, 5.3, and 6) [46].

The obsoleted TOS processing rules in TCP assumed bidirectional (or symmetric) precedence values used on a connection, but the DiffServ architecture is asymmetric. Problems with the old TCP logic in this regard were described in [20] and the solution described is to ignore IP precedence in TCP. Since RFC 2873 is a Standards Track document (although not marked as updating RFC 793), current implementations are expected to be robust to these conditions. Note that the DiffServ field value used in each direction is a part of the interface between TCP and the network layer, and values in use can be indicated both ways between TCP and the application.

A.1.2. MLS Systems

The IP security option (IPSO) and compartment defined in [1] was refined in RFC 1038 that was later obsoleted by RFC 1108. The Commercial IP Security Option (CIPSO) is defined in FIPS-188, and is supported by some vendors and operating systems. RFC 1108 is now Historic, though RFC 791 itself has not been updated to remove the IP security option. For IPv6, a similar option (CALIPSO) has been defined [30]. RFC 793 includes logic that includes the IP security/compartment information in treatment of TCP segments. References to the IP "security/compartment" in this document may be relevant for Multi-Level Secure (MLS) system implementers, but can be ignored for non-MLS implementations, consistent with running code on the Internet. See Appendix A.1 for further discussion. Note that RFC 5570 describes some MLS networking scenarios where IPSO, CIPSO, or CALIPSO may be used. In these special cases, TCP implementers should see section 7.3.1 of RFC 5570, and follow the guidance in that document.

A.2. Sequence Number Validation

There are cases where the TCP sequence number validation rules can prevent ACK fields from being processed. This can result in connection issues, as described in [55], which includes descriptions of potential problems in conditions of simultaneous open, self-connects, simultaneous close, and simultaneous window probes. The document also describes potential changes to the TCP specification to mitigate the issue by expanding the acceptable sequence numbers.

In Internet usage of TCP, these conditions are rarely occurring. Common operating systems include different alternative mitigations, and the standard has not been updated yet to codify one of them, but implementers should consider the problems described in [55].

A.3. Nagle Modification

In common operating systems, both the Nagle algorithm and delayed acknowledgements are implemented and enabled by default. TCP is used by many applications that have a request-response style of communication, where the combination of the Nagle algorithm and delayed acknowledgements can result in poor application performance. A modification to the Nagle algorithm is described in [58] that improves the situation for these applications.

This modification is implemented in some common operating systems, and does not impact TCP interoperability. Additionally, many applications simply disable Nagle, since this is generally supported by a socket option. The TCP standard has not been updated to include this Nagle modification, but implementers may find it beneficial to consider.

A.4. Low Water Mark Settings

Some operating system kernel TCP implementations include socket options that allow specifying the number of bytes in the buffer until the socket layer will pass sent data to TCP (SO_SNDLOWAT) or to the application on receiving (SO_RCVLOWAT).

In addition, another socket option (TCP_NOTSENT_LOWAT) can be used to control the amount of unsent bytes in the write queue. This can help a sending TCP application to avoid creating large amounts of buffered data (and corresponding latency). As an example, this may be useful for applications that are multiplexing data from multiple upper level streams onto a connection, especially when streams may be a mix of interactive / real-time and bulk data transfer.

Appendix B. TCP Requirement Summary

This section is adapted from RFC 1122.

Note that there is no requirement related to PLPMTUD in this list, but that PLPMTUD is recommended.

FEATURE	ReqID	MUST	SHOULD	MAY	SHOULD NOT	MUST NOT	Footnote
Push flag							
Aggregate or queue un-pushed data	MAY-16			x			
Sender collapse successive PSH flags	SHLD-27		x				
SEND call can specify PUSH	MAY-15			x			
If cannot: sender buffer indefinitely	MUST-60					x	
If cannot: PSH last segment	MUST-61	x					
Notify receiving ALP of PSH	MAY-17			x			1
Send max size segment when possible	SHLD-28		x				
Window							
Treat as unsigned number	MUST-1	x					
Handle as 32-bit number	REC-1		x				
Shrink window from right	SHLD-14				x		
- Send new data when window shrinks	SHLD-15				x		
- Retransmit old unacked data within window	SHLD-16		x				
- Time out conn for data past right edge	SHLD-17				x		
Robust against shrinking window	MUST-34	x					
Receiver’s window closed indefinitely	MAY-8			x			
Use standard probing logic	MUST-35	x					
Sender probe zero window	MUST-36	x					
First probe after RTO	SHLD-29		x				
Exponential backoff	SHLD-30		x				
Allow window stay zero indefinitely	MUST-37	x					
Retransmit old data beyond SND.UNA+SND.WND	MAY-7			x			
Process RST and URG even with zero window	MUST-66	x					
Urgent Data							
Include support for urgent pointer	MUST-30	x					

Pointer indicates first non-urgent octet	MUST-62	x					
Arbitrary length urgent data sequence	MUST-31	x					
Inform ALP asynchronously of urgent data	MUST-32	x					1
ALP can learn if/how much urgent data Q'd	MUST-33	x					1
ALP employ the urgent mechanism	SHLD-13				x		
TCP Options							
Support the mandatory option set	MUST-4	x					
Receive TCP option in any segment	MUST-5	x					
Ignore unsupported options	MUST-6	x					
Cope with illegal option length	MUST-7	x					
Process options regardless of word alignment	MUST-64	x					
Implement sending & receiving MSS option	MUST-14	x					
IPv4 Send MSS option unless 536	SHLD-5		x				
IPv6 Send MSS option unless 1220	SHLD-5		x				
Send MSS option always	MAY-3			x			
IPv4 Send-MSS default is 536	MUST-15	x					
IPv6 Send-MSS default is 1220	MUST-15	x					
Calculate effective send seg size	MUST-16	x					
MSS accounts for varying MTU	SHLD-6		x				
MSS not sent on non-SYN segments	MUST-65					x	
MSS value based on MMS_R	MUST-67	x					
TCP Checksums							
Sender compute checksum	MUST-2	x					
Receiver check checksum	MUST-3	x					
ISN Selection							
Include a clock-driven ISN generator component	MUST-8	x					
Secure ISN generator with a PRF component	SHLD-1		x				
PRF computable from outside the host	MUST-9					x	
Opening Connections							
Support simultaneous open attempts	MUST-10	x					
SYN-RECEIVED remembers last state	MUST-11	x					
Passive Open call interfere with others	MUST-41					x	
Function: simultan. LISTENs for same port	MUST-42	x					
Ask IP for src address for SYN if necc.	MUST-44	x					
Otherwise, use local addr of conn.	MUST-45	x					
OPEN to broadcast/multicast IP Address	MUST-46					x	
Silently discard seg to bcast/mcast addr	MUST-57	x					
Closing Connections							
RST can contain data	SHLD-2		x				
Inform application of aborted conn	MUST-12	x					
Half-duplex close connections	MAY-1			x			
Send RST to indicate data lost	SHLD-3		x				
In TIME-WAIT state for 2MSL seconds	MUST-13	x					

Accept SYN from TIME-WAIT state	MAY-2			x			
Use Timestamps to reduce TIME-WAIT	SHLD-4		x				
Retransmissions							
Implement RFC 5681	MUST-19	x					
Retransmit with same IP ident	MAY-4			x			
Karn’s algorithm	MUST-18	x					
Generating ACKs:							
Aggregate whenever possible	MUST-58	x					
Queue out-of-order segments	SHLD-31		x				
Process all Q’d before send ACK	MUST-59	x					
Send ACK for out-of-order segment	MAY-13			x			
Delayed ACKs	SHLD-18		x				
Delay < 0.5 seconds	MUST-40	x					
Every 2nd full-sized segment ACK’d	SHLD-19	x					
Receiver SWS-Avoidance Algorithm	MUST-39	x					
Sending data							
Configurable TTL	MUST-49	x					
Sender SWS-Avoidance Algorithm	MUST-38	x					
Nagle algorithm	SHLD-7		x				
Application can disable Nagle algorithm	MUST-17	x					
Connection Failures:							
Negative advice to IP on R1 retxs	MUST-20	x					
Close connection on R2 retxs	MUST-20	x					
ALP can set R2	MUST-21	x					1
Inform ALP of R1<=retxs<R2	SHLD-9		x				1
Recommended value for R1	SHLD-10		x				
Recommended value for R2	SHLD-11		x				
Same mechanism for SYNs	MUST-22	x					
R2 at least 3 minutes for SYN	MUST-23	x					
Send Keep-alive Packets:	MAY-5			x			
- Application can request	MUST-24	x					
- Default is "off"	MUST-25	x					
- Only send if idle for interval	MUST-26	x					
- Interval configurable	MUST-27	x					
- Default at least 2 hrs.	MUST-28	x					
- Tolerant of lost ACKs	MUST-29	x					
- Send with no data	SHLD-12		x				
- Configurable to send garbage octet	MAY-6			x			
IP Options							
Ignore options TCP doesn’t understand	MUST-50	x					
Time Stamp support	MAY-10			x			
Record Route support	MAY-11			x			

Source Route:							
ALP can specify	MUST-51	x					1
Overrides src rt in datagram	MUST-52	x					
Build return route from src rt	MUST-53	x					
Later src route overrides	SHLD-24		x				
Receiving ICMP Messages from IP	MUST-54	x					
Dest. Unreach (0,1,5) => inform ALP	SHLD-25		x				
Dest. Unreach (0,1,5) => abort conn	MUST-56					x	
Dest. Unreach (2-4) => abort conn	SHLD-26		x				
Source Quench => silent discard	MUST-55	x					
Time Exceeded => tell ALP, don't abort	MUST-56					x	
Param Problem => tell ALP, don't abort	MUST-56					x	
Address Validation							
Reject OPEN call to invalid IP address	MUST-46	x					
Reject SYN from invalid IP address	MUST-63	x					
Silently discard SYN to bcast/mcast addr	MUST-57	x					
TCP/ALP Interface Services							
Error Report mechanism	MUST-47	x					
ALP can disable Error Report Routine	SHLD-20		x				
ALP can specify DiffServ field for sending	MUST-48	x					
Passed unchanged to IP	SHLD-22		x				
ALP can change DiffServ field during connection	SHLD-21		x				
ALP generally changing DiffServ during conn.	SHLD-23				x		
Pass received DiffServ field up to ALP	MAY-9			x			
FLUSH call	MAY-14			x			
Optional local IP addr parm. in OPEN	MUST-43	x					
RFC 5961 Support:							
Implement data injection protection	MAY-12			x			
Explicit Congestion Notification:							
Support ECN	SHLD-8		x				

FOOTNOTES: (1) "ALP" means Application-Layer Program.

Author's Address

Wesley M. Eddy (editor)
MTI Systems
US

Email: wes@mti-systems.com

TCPM M. Scharf
Internet-Draft Hochschule Esslingen
Intended status: Standards Track V. Murgai
Expires: May 2, 2021

M. Jethanandani
Kloud Services
October 29, 2020

YANG Model for Transmission Control Protocol (TCP) Configuration
draft-ietf-tcpm-yang-tcp-00

Abstract

This document specifies a YANG model for TCP on devices that are configured by network management protocols. The YANG model defines a container for all TCP connections and groupings of some of the parameters that can be imported and used in TCP implementations or by other models that need to configure TCP parameters. The model includes definitions from YANG Groupings for TCP Client and TCP Servers (I-D.ietf-netconf-tcp-client-server). The model is NMDA (RFC 8342) compliant.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Transmission Control Protocol (TCP) [RFC0793] is used by many applications in the Internet, including control and management protocols. Therefore, TCP is implemented on network elements that can be configured via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. This document specifies a YANG [RFC7950] 1.1 model for configuring TCP on network elements that support YANG data models, and is Network Management Datastore Architecture (NMDA) [RFC8342] compliant. This module defines a container for TCP connection, and includes definitions from YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. The model has a narrow scope and focuses on fundamental TCP functions and basic statistics. The model can be augmented or updated to address more advanced or implementation-specific TCP features in the future.

Many protocol stacks on Internet hosts use other methods to configure TCP, such as operating system configuration or policies. Many TCP/IP stacks cannot be configured by network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. Moreover, many existing TCP/IP stacks do not use YANG data models. Such TCP implementations often have other means to configure the parameters listed in this document, which are outside the scope of this document.

This specification is orthogonal to the Management Information Base (MIB) for the Transmission Control Protocol (TCP) [RFC4022]. The basic statistics defined in this document follow the model of the TCP MIB. An TCP Extended Statistics MIB [RFC4898] is also available, but this document does not cover such extended statistics. It is possible also to translate a MIB into a YANG model, for instance using Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules [RFC6643]. However, this approach is not used in this document, as such a translated model would not be up-to-date.

There are other existing TCP-related YANG models, which are othogonal to this specification. Examples are:

- o TCP header attributes are modeled in other models, such as YANG Data Model for Network Access Control Lists (ACLs) [RFC8519] and Distributed Denial-of-Service Open Thread Signaling (DOTS) Data Channel Specification [I-D.ietf-dots-data-channel].

- o TCP-related configuration of a NAT (e.g., NAT44, NAT64, Destination NAT, ...) is defined in A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT) [RFC8512] and A YANG Data Model for Dual-Stack Lite (DS-Lite) [RFC8513].

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Note to RFC Editor

This document uses several placeholder values throughout the document. Please replace them as follows and remove this note before publication.

RFC XXXX, where XXXX is the number assigned to this document at the time of publication.

2020-10-29 with the actual date of the publication of this document.

3. Model Overview

3.1. Modeling Scope

TCP is implemented on many different system architectures. As a result, there are may different and often implementation-specific ways to configure parameters of the TCP protocol engine. In addition, in many TCP/IP stacks configuration exists for different scopes:

- Global configuration: Many TCP implementations have configuration parameters that affect all TCP connections. Typical examples include enabling or disabling optional protocol features.

- Interface configuration: It can be useful to use different TCP parameters on different interfaces, e.g., different device ports or IP interfaces. In that case, TCP parameters can be part of the interface configuration. Typical examples are the Maximum Segment Size (MSS) or configuration related to hardware offloading.

- Connection parameters: Many implementations have means to influence the behavior of each TCP connection, e.g., on the programming interface used by applications. A typical example are socket options in the socket API, such as disabling the Nagle algorithm by TCP_NODELAY. If an application uses such an interface, it is possible that the configuration of the application or application protocol includes TCP-related parameters. An example is the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].

- Policies: Setting of TCP parameters can also be part of system policies, templates, or profiles. An example would be the preferences defined in An Abstract Application Layer Interface to Transport Services [I-D.ietf-taps-interface].

As a result, there is no ground truth for setting certain TCP parameters, and traditionally different TCP implementation have used different modeling approaches. For instance, one implementation may define a given configuration parameter globally, while another one uses per-interface settings, and both approaches work well for the corresponding use cases. Also, different systems may use different default values. In addition, TCP can be implemented in different ways and design choices by the protocol engine often affect configuration options.

Nonetheless, a number of TCP stack parameters require configuration by YANG models. This document therefore defines a minimal YANG model with fundamental parameters directly following from TCP standards.

An important use case is the TCP configuration on network elements such as routers, which often use YANG data models. The model therefore specifies TCP parameters that are important on such TCP stacks. A typical example is the support of TCP-AO [RFC5925]. TCP-AO is increasingly supported on routers to secure routing protocols such as BGP. In that case, TCP-AO configuration is required on routers.

Given an installed base, the model also allows enabling of the legacy TCP MD5 [RFC2385] signature option. As the TCP MD5 signature option is obsoleted by TCP-AO, it is strongly RECOMMENDED to use TCP-AO.

Similar to the TCP MIB [RFC4022], this document also specifies basic statistics and a TCP connection table.

- Statistics: Counters for the number of active/passive opens, sent and received segments, errors, and possibly other detailed debugging information

- TCP connection table: Access to status information for all TCP connections

This allows implementations of TCP MIB [RFC4022] to migrate to the YANG model defined in this memo.

3.2. Model Design

The YANG model defined in this document includes definitions from the YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. Similar to that model, this specification defines YANG groupings. This allows reuse of these groupings in different YANG data models. It is intended that these groupings will be used either standalone or for TCP-based protocols as part of a stack of protocol-specific configuration models. An example could be the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].

3.3. Tree Diagram

This section provides a abridged tree diagram for the YANG module defined in this document. Annotations used in the diagram are defined in YANG Tree Diagrams [RFC8340].

```
   module: ietf-tcp
     +--rw tcp!
        +--rw connections
        |     ...
        +--rw server {server}?
        |     ...
        +--rw client {client}?
        |     ...
        +--ro statistics {statistics}?
              ...
```

4. TCP YANG Model

```
  <CODE BEGINS> file "ietf-tcp@2020-10-29.yang"

  module ietf-tcp {
    yang-version "1.1";
    namespace "urn:ietf:params:xml:ns:yang:ietf-tcp";
    prefix "tcp";

    import ietf-yang-types {
      prefix "yang";
      reference
        "RFC 6991: Common YANG Data Types.";
    }
    import ietf-tcp-client {
      prefix "tcpc";
    }
    import ietf-tcp-server {
      prefix "tcps";
    }
    import ietf-tcp-common {
      prefix "tcpcmn";
    }
    import ietf-inet-types {
      prefix "inet";
    }

    organization
      "IETF TCPM Working Group";

    contact
```

```
      "WG Web:   <http://tools.ietf.org/wg/tcpm>
       WG List:  <tcpm@ietf.org>

       Authors: Michael Scharf (michael.scharf at hs-esslingen dot de)
                Vishal Murgai (vmurgai at gmail dot com)
                Mahesh Jethanandani (mjethanandani at gmail dot com)";

    description
      "This module focuses on fundamental and standard TCP functions
       that widely implemented. The model can be augmented to address
       more advanced or implementation specific TCP features.";

    revision "2020-10-29" {
      description
        "Initial Version";
      reference
        "RFC XXX, TCP Configuration.";
    }

    // Features
    feature server {
      description
        "TCP Server configuration supported.";
    }

    feature client {
      description
        "TCP Client configuration supported.";
    }

    feature statistics {
      description
        "This implementation supports statistics reporting.";
    }

    // TCP-AO Groupings

    grouping ao {
      leaf enable-ao {
        type boolean;
        default "false";
        description
          "Enable support of TCP-Authentication Option (TCP-AO).";
      }

      leaf send-id {
        type uint8 {
          range "0..255";
        }
        must "../enable-ao = ’true’";
        description
          "The SendID is inserted as the KeyID of the TCP-AO option
           of outgoing segments.";
        reference
          "RFC 5925: The TCP Authentication Option.";
      }

      leaf recv-id {
        type uint8 {
          range "0..255";
        }
        must "../enable-ao = ’true’";
        description
          "The RecvID is matched against the TCP-AO KeyID of incoming
           segments.";
        reference
          "RFC 5925: The TCP Authentication Option.";
      }

      leaf include-tcp-options {
        type boolean;
        must "../enable-ao = ’true’";
        description
          "Include TCP options in HMAC calculation.";
      }

      leaf accept-ao-mismatch {
        type boolean;
        must "../enable-ao = ’true’";
        description
          "Accept packets with HMAC mismatch.";
      }
      description
        "Authentication Option (AO) for TCP.";
      reference
        "RFC 5925: The TCP Authentication Option.";
    }

    // MD5 grouping

    grouping md5 {
      description
        "Grouping for use in authenticating TCP sessions using MD5.";
      reference
        "RFC 2385: Protection of BGP Sessions via the TCP MD5
         Signature.";
      leaf enable-md5 {
        type boolean;
        default "false";
        description
          "Enable support of MD5 to authenticate a TCP session.";
      }
    }

    // TCP configuration

    container tcp {
      presence "The container for TCP configuration.";

      description
        "TCP container.";

      container connections {
        list connection {
          key "local-address remote-address local-port remote-port";

          leaf local-address {
            type inet:ip-address;
            description
              "Local address that forms the connection identifier.";
          }

          leaf remote-address {
            type inet:ip-address;
            description
              "Remote address that forms the connection identifier.";
          }

          leaf local-port {
            type inet:port-number;
            description
              "Local TCP port that forms the connection identifier.";
          }

          leaf remote-port {
            type inet:port-number;
            description
              "Remote TCP port that forms the connection identifier.";
          }

          container common {
            uses tcpcmn:tcp-common-grouping;

            choice authentication {
              case ao {
                uses ao;
                description
                  "Use TCP-AO to secure the connection.";
              }

              case md5 {
                uses md5;
                description
                  "Use TCP-MD5 to secure the connection.";
              }
              description
                "Choice of how to secure the TCP connection.";
            }
            description
              "Common definitions of TCP configuration. This includes
               parameters such as how to secure the connection,
               that can be part of either the client or server.";
          }
          description
            "Connection related parameters.";
        }
        description
          "A container of all TCP connections.";
      }

      container server {
        if-feature server;
        uses tcps:tcp-server-grouping;
        description
          "Definitions of TCP server configuration.";
      }

      container client {
        if-feature client;
        uses tcpc:tcp-client-grouping;
        description
          "Definitions of TCP client configuration.";
      }

      container statistics {
        if-feature statistics;
        config false;

        leaf active-opens {
          type yang:counter32;
          description
            "The number of times that TCP connections have made a direct
```

```
             transition to the SYN-SENT state from the CLOSED state.";
        }

        leaf passive-opens {
          type yang:counter32;
          description
            "The number of times TCP connections have made a direct
             transition to the SYN-RCVD state from the LISTEN state.";
        }

        leaf attempt-fails {
          type yang:counter32;
          description
            "The number of times that TCP connections have made a direct
             transition to the CLOSED state from either the SYN-SENT
             state or the SYN-RCVD state, plus the number of times that
             TCP connections have made a direct transition to the
             LISTEN state from the SYN-RCVD state.";
        }

        leaf establish-resets {
          type yang:counter32;
          description
            "The number of times that TCP connections have made a direct
             transition to the CLOSED state from either the ESTABLISHED
             state or the CLOSE-WAIT state.";
        }

        leaf currently-established {
          type yang:gauge32;
          description
            "The number of TCP connections for which the current state
             is either ESTABLISHED or CLOSE-WAIT.";
        }

        leaf in-segments {
          type yang:counter64;
          description
            "The total number of segments received, including those
             received in error.  This count includes segments received
             on currently established connections.";
        }

        leaf out-segments {
          type yang:counter64;
          description
            "The total number of segments sent, including those on
             current connections but excluding those containing only
             retransmitted octets.";
        }

        leaf retransmitted-segments {
          type yang:counter32;
          description
            "The total number of segments retransmitted; that is, the
             number of TCP segments transmitted containing one or more
             previously transmitted octets.";
        }

        leaf in-errors {
          type yang:counter32;
          description
            "The total number of segments received in error (e.g., bad
             TCP checksums).";
        }

        leaf out-resets {
          type yang:counter32;
          description
            "The number of TCP segments sent containing the RST flag.";
        }

        action reset {
          description
            "Reset statistics action command.";
          input {
            leaf reset-at {
              type yang:date-and-time;
              description
                "Time when the reset action needs to be
                 executed.";
            }
          }
          output {
            leaf reset-finished-at {
              type yang:date-and-time;
              description
                "Time when the reset action command completed.";
            }
          }
        }
        description
          "Statistics across all connections.";
      }
    }
  }
  <CODE ENDS>
```

5. IANA Considerations

5.1. The IETF XML Registry

This document registers two URIs in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in IETF XML Registry [RFC3688], the following registrations are requested:

```
URI: urn:ietf:params:xml:ns:yang:ietf-tcp
Registrant Contact: The TCPM WG of the IETF.
XML: N/A, the requested URI is an XML namespace.
```

5.2. The YANG Module Names Registry

This document registers a YANG modules in the YANG Module Names registry YANG - A Data Modeling Language [RFC6020]. Following the format in YANG - A Data Modeling Language [RFC6020], the following registrations are requested:

```
name:         ietf-tcp
namespace:    urn:ietf:params:xml:ns:yang:ietf-tcp
prefix:       tcp
reference:    RFC XXXX
```

6. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) described in Using the NETCONF protocol over SSH [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

7. References

7.1. Normative References

[I-D.ietf-netconf-tcp-client-server]
Watsen, K. and M. Scharf, "YANG Groupings for TCP Clients and TCP Servers", draft-ietf-netconf-tcp-client-server-08 (work in progress), August 2020.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, DOI 10.17487/RFC2385, August 1998, <https://www.rfc-editor.org/info/rfc2385>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <https://www.rfc-editor.org/info/rfc5925>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

7.2. Informative References

[I-D.ietf-dots-data-channel] Boucadair, M. and T. Reddy.K, "Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification", draft-ietf-dots-data-channel-31 (work in progress), July 2019.

[I-D.ietf-idr-bgp-model] Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", draft-ietf-idr-bgp-model-09 (work in progress), June 2020.

[I-D.ietf-taps-interface] Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kuehlewind, M., Perkins, C., Tiesel, P., Wood, C., and T. Pauly, "An Abstract Application Layer Interface to Transport Services", draft-ietf-taps-interface-09 (work in progress), July 2020.

[RFC4022] Raghunarayan, R., Ed., "Management Information Base for the Transmission Control Protocol (TCP)", RFC 4022, DOI 10.17487/RFC4022, March 2005, <https://www.rfc-editor.org/info/rfc4022>.

[RFC4898] Mathis, M., Heffner, J., and R. Raghunarayan, "TCP Extended Statistics MIB", RFC 4898, DOI 10.17487/RFC4898, May 2007, <https://www.rfc-editor.org/info/rfc4898>.

[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <https://www.rfc-editor.org/info/rfc6643>.

[RFC8512] Boucadair, M., Ed., Sivakumar, S., Jacquenet, C., Vinapamula, S., and Q. Wu, "A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT)", RFC 8512, DOI 10.17487/RFC8512, January 2019, <https://www.rfc-editor.org/info/rfc8512>.

[RFC8513] Boucadair, M., Jacquenet, C., and S. Sivakumar, "A YANG Data Model for Dual-Stack Lite (DS-Lite)", RFC 8513, DOI 10.17487/RFC8513, January 2019, <https://www.rfc-editor.org/info/rfc8513>.

[RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <https://www.rfc-editor.org/info/rfc8519>.

Appendix A. Acknowledgements

Michael Scharf was supported by the StandICT.eu project, which is funded by the European Commission under the Horizon 2020 Programme.

The following persons have contributed to this document by reviews: Mohamed Boucadair

Appendix B. Changes compared to previous versions

Changes compared to draft-scharf-tcpm-yang-tcp-04

- Removed congestion control

- Removed global stack parameters

Changes compared to draft-scharf-tcpm-yang-tcp-03

- Updated TCP-AO grouping

- Added congestion control

Changes compared to draft-scharf-tcpm-yang-tcp-02

- Initial proposal of a YANG model including base configuration parameters, TCP-AO configuration, and a connection list

- Editorial bugfixes and outdated references reported by Mohamed Boucadair

- Additional co-author Mahesh Jethanandani

Changes compared to draft-scharf-tcpm-yang-tcp-01

- Alignment with [I-D.ietf-netconf-tcp-client-server]

- Removing backward-compatibility to the TCP MIB

- Additional co-author Vishal Murgai

Changes compared to draft-scharf-tcpm-yang-tcp-00

- Editorial improvements

Appendix C. Examples

C.1. Keepalive Configuration

This particular example demonstrates how both a particular connection can be configured for keepalives.

```
   <?xml version="1.0" encoding="UTF-8"?>
   <!--
    This example shows how TCP keepalive can be configured for
    a given connection. An idle connection is dropped after
    idle-time + (max-probes * probe-interval).
   -->
   <config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
     <tcp
         xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
       <connections>
         <connection>
           <local-address>192.168.1.1</local-address>
           <remote-address>192.168.1.2</remote-address>
           <local-port>1025</local-port>
           <remote-port>80</remote-port>
           <common>
             <keepalives>
               <idle-time>5</idle-time>
               <max-probes>5</max-probes>
               <probe-interval>10</probe-interval>
             </keepalives>
           </common>
         </connection>
       </connections>
       <!--
        It is not clear why a server and client configuration is
        needed here even as they under a feature statement and therefore
        are required only if the feature is declared. Adding it so
        that yanglint allows this validation to run.
       -->
       <server>
         <local-address>192.168.1.1</local-address>
       </server>
       <client>
         <remote-address>192.168.1.2</remote-address>
       </client>
     </tcp>
   </config>
```

Authors' Addresses

Michael Scharf
Hochschule Esslingen - University of Applied Sciences
Flandernstr. 101
Esslingen 73732
Germany

Email: michael.scharf@hs-esslingen.de

Vishal Murgai

Email: vmurgai@gmail.com

Mahesh Jethanandani
Kloud Services

Email: mjethanandani@gmail.com

TCPM
Internet-Draft
Intended status: Standards Track
Expires: May 20, 2021

M. Scharf
Hochschule Esslingen
V. Murgai
Samsung
M. Jethanandani
Kloud Services
November 16, 2020

YANG Model for Transmission Control Protocol (TCP) Configuration
draft-ietf-tcpm-yang-tcp-01

Abstract

This document specifies a YANG model for TCP on devices that are configured by network management protocols. The YANG model defines a container for all TCP connections and groupings of some of the parameters that can be imported and used in TCP implementations or by other models that need to configure TCP parameters. The model includes definitions from YANG Groupings for TCP Client and TCP Servers (I-D.ietf-netconf-tcp-client-server). The model is NMDA (RFC 8342) compliant.

Status of This Memo

This Internet-Draft will expire on May 20, 2021.

Copyright Notice

Table of Contents

1. Introduction

The Transmission Control Protocol (TCP) [RFC0793] is used by many applications in the Internet, including control and management protocols. Therefore, TCP is implemented on network elements that can be configured via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. This document specifies a YANG [RFC7950] 1.1 model for configuring TCP on network elements that support YANG data models, and is Network Management Datastore Architecture (NMDA) [RFC8342] compliant. This module defines a container for TCP connection, and includes definitions from YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. The model has a narrow scope and focuses on fundamental TCP functions and basic statistics. The model can be augmented or updated to address more advanced or implementation-specific TCP features in the future.

Many protocol stacks on Internet hosts use other methods to configure TCP, such as operating system configuration or policies. Many TCP/IP stacks cannot be configured by network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. Moreover, many existing TCP/IP stacks do not use YANG data models. Such TCP implementations often have other means to configure the parameters listed in this document, which are outside the scope of this document.

This specification is orthogonal to the Management Information Base (MIB) for the Transmission Control Protocol (TCP) [RFC4022]. The basic statistics defined in this document follow the model of the TCP MIB. An TCP Extended Statistics MIB [RFC4898] is also available, but this document does not cover such extended statistics. It is possible also to translate a MIB into a YANG model, for instance using Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules [RFC6643]. However, this approach is not used in this document, as such a translated model would not be up-to-date.

There are other existing TCP-related YANG models, which are orthogonal to this specification. Examples are:

- TCP header attributes are modeled in other models, such as YANG Data Model for Network Access Control Lists (ACLs) [RFC8519] and Distributed Denial-of-Service Open Thread Signaling (DOTS) Data Channel Specification [RFC8783].

- TCP-related configuration of a NAT (e.g., NAT44, NAT64, Destination NAT, ...) is defined in A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT) [RFC8512] and A YANG Data Model for Dual-Stack Lite (DS-Lite) [RFC8513].

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.1. Note to RFC Editor

This document uses several placeholder values throughout the document. Please replace them as follows and remove this note before publication.

RFC XXXX, where XXXX is the number assigned to this document at the time of publication.

2020-11-16 with the actual date of the publication of this document.

3. Model Overview

3.1. Modeling Scope

TCP is implemented on many different system architectures. As a result, there are may different and often implementation-specific ways to configure parameters of the TCP protocol engine. In addition, in many TCP/IP stacks configuration exists for different scopes:

- Global configuration: Many TCP implementations have configuration parameters that affect all TCP connections. Typical examples include enabling or disabling optional protocol features.

- Interface configuration: It can be useful to use different TCP parameters on different interfaces, e.g., different device ports or IP interfaces. In that case, TCP parameters can be part of the interface configuration. Typical examples are the Maximum Segment Size (MSS) or configuration related to hardware offloading.

- Connection parameters: Many implementations have means to influence the behavior of each TCP connection, e.g., on the programming interface used by applications. A typical example are socket options in the socket API, such as disabling the Nagle algorithm by TCP_NODELAY. If an application uses such an interface, it is possible that the configuration of the application or application protocol includes TCP-related parameters. An example is the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].

- Policies: Setting of TCP parameters can also be part of system policies, templates, or profiles. An example would be the preferences defined in An Abstract Application Layer Interface to Transport Services [I-D.ietf-taps-interface].

As a result, there is no ground truth for setting certain TCP parameters, and traditionally different TCP implementation have used different modeling approaches. For instance, one implementation may define a given configuration parameter globally, while another one uses per-interface settings, and both approaches work well for the corresponding use cases. Also, different systems may use different default values. In addition, TCP can be implemented in different ways and design choices by the protocol engine often affect configuration options.

Nonetheless, a number of TCP stack parameters require configuration by YANG models. This document therefore defines a minimal YANG model with fundamental parameters directly following from TCP standards.

An important use case is the TCP configuration on network elements such as routers, which often use YANG data models. The model therefore specifies TCP parameters that are important on such TCP stacks.

A typical example is the support of TCP-AO [RFC5925]. TCP-AO is increasingly supported on routers to secure routing protocols such as BGP. In that case, TCP-AO configuration is required on routers. The model includes the required TCP parameters for TCP-AO configuration. The key chain for TCP-AO can be modeled by the YANG Data Model for Key Chains [RFC8177].

Given an installed base, the model also allows enabling of the legacy TCP MD5 [RFC2385] signature option. As the TCP MD5 signature option is obsoleted by TCP-AO, it is strongly RECOMMENDED to use TCP-AO.

Similar to the TCP MIB [RFC4022], this document also specifies basic statistics and a TCP connection table.

- Statistics: Counters for the number of active/passive opens, sent and received segments, errors, and possibly other detailed debugging information

- TCP connection table: Access to status information for all TCP connections

This allows implementations of TCP MIB [RFC4022] to migrate to the YANG model defined in this memo.

3.2. Model Design

The YANG model defined in this document includes definitions from the YANG Groupings for TCP Clients and TCP Servers [I-D.ietf-netconf-tcp-client-server]. Similar to that model, this specification defines YANG groupings. This allows reuse of these groupings in different YANG data models. It is intended that these groupings will be used either standalone or for TCP-based protocols as part of a stack of protocol-specific configuration models. An example could be the BGP YANG Model for Service Provider Networks [I-D.ietf-idr-bgp-model].

3.3. Tree Diagram

This section provides a abridged tree diagram for the YANG module defined in this document. Annotations used in the diagram are defined in YANG Tree Diagrams [RFC8340].

```
   module: ietf-tcp
     +--rw tcp!
        +--rw connections
        |     ...
        +--rw server {server}?
        |     ...
        +--rw client {client}?
        |     ...
        +--ro statistics {statistics}?
              ...
```

4. TCP YANG Model

```
  <CODE BEGINS> file "ietf-tcp@2020-11-16.yang"

  module ietf-tcp {
    yang-version "1.1";
    namespace "urn:ietf:params:xml:ns:yang:ietf-tcp";
    prefix "tcp";

    import ietf-yang-types {
      prefix "yang";
      reference
        "RFC 6991: Common YANG Data Types.";
    }
    import ietf-tcp-client {
      prefix "tcpc";
    }
    import ietf-tcp-server {
      prefix "tcps";
    }
    import ietf-tcp-common {
      prefix "tcpcmn";
    }
    import ietf-inet-types {
      prefix "inet";
    }

    organization
      "IETF TCPM Working Group";

    contact
```

```
      "WG Web:   <http://tools.ietf.org/wg/tcpm>
       WG List:  <tcpm@ietf.org>

       Authors: Michael Scharf (michael.scharf at hs-esslingen dot de)
                Vishal Murgai (vmurgai at gmail dot com)
                Mahesh Jethanandani (mjethanandani at gmail dot com)";

    description
      "This module focuses on fundamental and standard TCP functions
       that widely implemented. The model can be augmented to address
       more advanced or implementation specific TCP features.

       Copyright (c) 2020 IETF Trust and the persons identified as
       authors of the code.  All rights reserved.

       Redistribution and use in source and binary forms, with or
       without modification, is permitted pursuant to, and subject to
       the license terms contained in, the Simplified BSD License set
       forth in Section 4.c of the IETF Trust's Legal Provisions
       Relating to IETF Documents
       (https://trustee.ietf.org/license-info).

       This version of this YANG module is part of RFC XXXX
       (https://www.rfc-editor.org/info/rfcXXXX); see the RFC itself
       for full legal notices.

       The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
       NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
       'MAY', and 'OPTIONAL' in this document are to be interpreted as
       described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
       they appear in all capitals, as shown here.";

    revision "2020-11-16" {
      description
        "Initial Version";
      reference
        "RFC XXXX, TCP Configuration.";
    }

    // Features
    feature server {
      description
        "TCP Server configuration supported.";
    }

    feature client {
      description
        "TCP Client configuration supported.";
    }

    feature statistics {
      description
        "This implementation supports statistics reporting.";
    }

    // TCP-AO Groupings

    grouping ao {
      leaf enable-ao {
        type boolean;
        default "false";
        description
          "Enable support of TCP-Authentication Option (TCP-AO).";
      }

      leaf send-id {
        type uint8 {
          range "0..255";
        }
        must "../enable-ao = 'true'";
        description
          "The SendID is inserted as the KeyID of the TCP-AO option
           of outgoing segments.";
        reference
          "RFC 5925: The TCP Authentication Option.";
      }

      leaf recv-id {
        type uint8 {
          range "0..255";
        }
        must "../enable-ao = 'true'";
        description
          "The RecvID is matched against the TCP-AO KeyID of incoming
           segments.";
        reference
          "RFC 5925: The TCP Authentication Option.";
      }

      leaf include-tcp-options {
        type boolean;
        must "../enable-ao = 'true'";
        default true;
        description
          "Include TCP options in MAC calculation.";
      }
      leaf accept-key-mismatch {
        type boolean;
        must "../enable-ao = ’true’";
        description
          "Accept TCP segments with a Master Key Tuple (MKT) that is not
           configured.";
      }
      description
        "Authentication Option (AO) for TCP.";
      reference
        "RFC 5925: The TCP Authentication Option.";
    }

    // MD5 grouping

    grouping md5 {
      description
        "Grouping for use in authenticating TCP sessions using MD5.";
      reference
        "RFC 2385: Protection of BGP Sessions via the TCP MD5
         Signature.";

      leaf enable-md5 {
        type boolean;
        default "false";
        description
          "Enable support of MD5 to authenticate a TCP session.";
      }
    }

    // TCP configuration

    container tcp {
      presence "The container for TCP configuration.";

      description
        "TCP container.";

      container connections {
        list connection {
          key "local-address remote-address local-port remote-port";

          leaf local-address {
            type inet:ip-address;
            description
              "Local address that forms the connection identifier.";
          }
          leaf remote-address {
            type inet:ip-address;
            description
              "Remote address that forms the connection identifier.";
          }

          leaf local-port {
            type inet:port-number;
            description
              "Local TCP port that forms the connection identifier.";
          }

          leaf remote-port {
            type inet:port-number;
            description
              "Remote TCP port that forms the connection identifier.";
          }

          container common {
            uses tcpcmn:tcp-common-grouping;

            choice authentication {
              case ao {
                uses ao;
                description
                  "Use TCP-AO to secure the connection.";
              }

              case md5 {
                uses md5;
                description
                  "Use TCP-MD5 to secure the connection.";
              }
              description
                "Choice of how to secure the TCP connection.";
            }
            description
              "Common definitions of TCP configuration. This includes
               parameters such as how to secure the connection,
               that can be part of either the client or server.";
          }
          description
            "Connection related parameters.";
        }
        description
          "A container of all TCP connections.";
      }
      container server {
        if-feature server;
        uses tcps:tcp-server-grouping;
        description
          "Definitions of TCP server configuration.";
      }

      container client {
        if-feature client;
        uses tcpc:tcp-client-grouping;
        description
          "Definitions of TCP client configuration.";
      }

      container statistics {
        if-feature statistics;
        config false;

        leaf active-opens {
          type yang:counter32;
          description
            "The number of times that TCP connections have made a direct
             transition to the SYN-SENT state from the CLOSED state.";
        }

        leaf passive-opens {
          type yang:counter32;
          description
            "The number of times TCP connections have made a direct
             transition to the SYN-RCVD state from the LISTEN state.";
        }

        leaf attempt-fails {
          type yang:counter32;
          description
            "The number of times that TCP connections have made a direct
             transition to the CLOSED state from either the SYN-SENT
             state or the SYN-RCVD state, plus the number of times that
             TCP connections have made a direct transition to the
             LISTEN state from the SYN-RCVD state.";
        }

        leaf establish-resets {
          type yang:counter32;
          description
            "The number of times that TCP connections have made a direct
             transition to the CLOSED state from either the ESTABLISHED
             state or the CLOSE-WAIT state.";
        }

        leaf currently-established {
          type yang:gauge32;
          description
            "The number of TCP connections for which the current state
             is either ESTABLISHED or CLOSE-WAIT.";
        }

        leaf in-segments {
          type yang:counter64;
          description
            "The total number of segments received, including those
             received in error.  This count includes segments received
             on currently established connections.";
        }

        leaf out-segments {
          type yang:counter64;
          description
            "The total number of segments sent, including those on
             current connections but excluding those containing only
             retransmitted octets.";
        }

        leaf retransmitted-segments {
          type yang:counter32;
          description
            "The total number of segments retransmitted; that is, the
             number of TCP segments transmitted containing one or more
             previously transmitted octets.";
        }

        leaf in-errors {
          type yang:counter32;
          description
            "The total number of segments received in error (e.g., bad
             TCP checksums).";
        }

        leaf out-resets {
          type yang:counter32;
          description
            "The number of TCP segments sent containing the RST flag.";
        }

        action reset {
          description
```

```
            "Reset statistics action command.";
          input {
            leaf reset-at {
              type yang:date-and-time;
              description
                "Time when the reset action needs to be
                 executed.";
            }
          }
          output {
            leaf reset-finished-at {
              type yang:date-and-time;
              description
                "Time when the reset action command completed.";
            }
          }
        }
        description
          "Statistics across all connections.";
      }
    }
  }

  <CODE ENDS>
```

5. IANA Considerations

5.1. The IETF XML Registry

This document registers two URIs in the "ns" subregistry of the IETF XML Registry [RFC3688]. Following the format in IETF XML Registry [RFC3688], the following registrations are requested:

URI: urn:ietf:params:xml:ns:yang:ietf-tcp
Registrant Contact: The TCPM WG of the IETF.
XML: N/A, the requested URI is an XML namespace.

5.2. The YANG Module Names Registry

This document registers a YANG modules in the YANG Module Names registry YANG - A Data Modeling Language [RFC6020]. Following the format in YANG - A Data Modeling Language [RFC6020], the following registrations are requested:

```
name:         ietf-tcp
namespace:    urn:ietf:params:xml:ns:yang:ietf-tcp
prefix:       tcp
reference:    RFC XXXX
```

6. Security Considerations

The YANG module specified in this document defines a schema for data that is designed to be accessed via network management protocols such as NETCONF [RFC6241] or RESTCONF [RFC8040]. The lowest NETCONF layer is the secure transport layer, and the mandatory-to-implement secure transport is Secure Shell (SSH) described in Using the NETCONF protocol over SSH [RFC6242]. The lowest RESTCONF layer is HTTPS, and the mandatory-to-implement secure transport is TLS [RFC8446].

The Network Configuration Access Control Model (NACM) [RFC8341] provides the means to restrict access for particular NETCONF or RESTCONF users to a preconfigured subset of all available NETCONF or RESTCONF protocol operations and content.

There are a number of data nodes defined in this YANG module that are writable/creatable/deletable (i.e., "config true", which is the default). These data nodes may be considered sensitive or vulnerable in some network environments. Write operations (e.g., edit-config) to these data nodes without proper protection can have a negative effect on network operations. These are the subtrees and data nodes and their sensitivity/vulnerability:

- Server configuration. Unrestricted access to all the nodes under server configuration, e.g. local-address or keepalive idle-timer, can cause connections to the server to fail or to timeout prematurely.

- Client configuration. Similar to server configuration, unrestricted access to the nodes under client configuration can cause connections from the client to fail, or connections to the server to be redirected, and in case of keepalive, cause connections to timeout prematurely etc.

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access (e.g., via get, get-config, or notification) to these data nodes. These are the subtrees and data nodes and their sensitivity/vulnerability:

- Unrestricted access to connection information of the client or server can be used by a malicious user to launch an attack, e.g. MITM.

- Similarly, unrestricted access to statistics of the client or server can be used by a malicious user to exploit any vulnerabilities of the system.

Some of the RPC operations in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control access to these operations. These are the operations and their sensitivity/vulnerability:

- The YANG module allows for the statistics to be cleared by executing the reset action. This action should be restricted to users with the right permission.

7. References

7.1. Normative References

[I-D.ietf-netconf-tcp-client-server]
Watsen, K. and M. Scharf, "YANG Groupings for TCP Clients and TCP Servers", draft-ietf-netconf-tcp-client-server-08 (work in progress), August 2020.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, DOI 10.17487/RFC2385, August 1998, <https://www.rfc-editor.org/info/rfc2385>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, DOI 10.17487/RFC5925, June 2010, <https://www.rfc-editor.org/info/rfc5925>.

[RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6242] Wasserman, M., "Using the NETCONF Protocol over Secure Shell (SSH)", RFC 6242, DOI 10.17487/RFC6242, June 2011, <https://www.rfc-editor.org/info/rfc6242>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8177] Lindem, A., Ed., Qu, Y., Yeung, D., Chen, I., and J. Zhang, "YANG Data Model for Key Chains", RFC 8177, DOI 10.17487/RFC8177, June 2017, <https://www.rfc-editor.org/info/rfc8177>.

[RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <https://www.rfc-editor.org/info/rfc8342>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

7.2. Informative References

[I-D.ietf-idr-bgp-model] Jethanandani, M., Patel, K., Hares, S., and J. Haas, "BGP YANG Model for Service Provider Networks", draft-ietf-idr-bgp-model-09 (work in progress), June 2020.

[I-D.ietf-taps-interface]
Trammell, B., Welzl, M., Enghardt, T., Fairhurst, G., Kuehlewind, M., Perkins, C., Tiesel, P., Wood, C., and T. Pauly, "An Abstract Application Layer Interface to Transport Services", draft-ietf-taps-interface-10 (work in progress), November 2020.

[I-D.touch-tcpm-ao-test-vectors]
Touch, J. and J. Kuusisaari, "TCP-AO Test Vectors", draft-touch-tcpm-ao-test-vectors-01 (work in progress), August 2020.

[RFC4022] Raghunarayan, R., Ed., "Management Information Base for the Transmission Control Protocol (TCP)", RFC 4022, DOI 10.17487/RFC4022, March 2005, <https://www.rfc-editor.org/info/rfc4022>.

[RFC4898] Mathis, M., Heffner, J., and R. Raghunarayan, "TCP Extended Statistics MIB", RFC 4898, DOI 10.17487/RFC4898, May 2007, <https://www.rfc-editor.org/info/rfc4898>.

[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <https://www.rfc-editor.org/info/rfc6643>.

[RFC8512] Boucadair, M., Ed., Sivakumar, S., Jacquenet, C., Vinapamula, S., and Q. Wu, "A YANG Module for Network Address Translation (NAT) and Network Prefix Translation (NPT)", RFC 8512, DOI 10.17487/RFC8512, January 2019, <https://www.rfc-editor.org/info/rfc8512>.

[RFC8513] Boucadair, M., Jacquenet, C., and S. Sivakumar, "A YANG Data Model for Dual-Stack Lite (DS-Lite)", RFC 8513, DOI 10.17487/RFC8513, January 2019, <https://www.rfc-editor.org/info/rfc8513>.

[RFC8519] Jethanandani, M., Agarwal, S., Huang, L., and D. Blair, "YANG Data Model for Network Access Control Lists (ACLs)", RFC 8519, DOI 10.17487/RFC8519, March 2019, <https://www.rfc-editor.org/info/rfc8519>.

[RFC8783] Boucadair, M., Ed. and T. Reddy.K, Ed., "Distributed Denial-of-Service Open Threat Signaling (DOTS) Data Channel Specification", RFC 8783, DOI 10.17487/RFC8783, May 2020, <https://www.rfc-editor.org/info/rfc8783>.

Appendix A. Acknowledgements

Michael Scharf was supported by the StandICT.eu project, which is funded by the European Commission under the Horizon 2020 Programme.

The following persons have contributed to this document by reviews: Mohamed Boucadair

Appendix B. Changes compared to previous versions

Changes compared to draft-scharf-tcpm-yang-tcp-04

o Removed congestion control

o Removed global stack parameters

Changes compared to draft-scharf-tcpm-yang-tcp-03

o Updated TCP-AO grouping

o Added congestion control

Changes compared to draft-scharf-tcpm-yang-tcp-02

o Initial proposal of a YANG model including base configuration parameters, TCP-AO configuration, and a connection list

o Editorial bugfixes and outdated references reported by Mohamed Boucadair

o Additional co-author Mahesh Jethanandani

Changes compared to draft-scharf-tcpm-yang-tcp-01

o Alignment with [I-D.ietf-netconf-tcp-client-server]

o Removing backward-compatibility to the TCP MIB

o Additional co-author Vishal Murgai

Changes compared to draft-scharf-tcpm-yang-tcp-00

o Editorial improvements

Appendix C. Examples

C.1. Keepalive Configuration

This particular example demonstrates how both a particular connection can be configured for keepalives.

[note: '\' line wrapping for formatting only]

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 This example shows how TCP keepalive can be configured for
 a given connection. An idle connection is dropped after
 idle-time + (max-probes * probe-interval).
-->
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <tcp
      xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
    <connections>
      <connection>
        <local-address>192.168.1.1</local-address>
        <remote-address>192.168.1.2</remote-address>
        <local-port>1025</local-port>
        <remote-port>80</remote-port>
        <common>
          <keepalives>
            <idle-time>5</idle-time>
            <max-probes>5</max-probes>
            <probe-interval>10</probe-interval>
          </keepalives>
        </common>
      </connection>
    </connections>
    <!--
     It is not clear why a server and client configuration is
     needed here even as they under a feature statement and
     therefore are required only if the feature is declared.
     Adding it so that yanglint allows this validation to run.
    -->
    <server>
      <local-address>192.168.1.1</local-address>
    </server>
    <client>
      <remote-address>192.168.1.2</remote-address>
    </client>
  </tcp>
</config>
```

C.2. TCP-AO Configuration

The following example demonstrates how to model a TCP-AO [RFC5925] configuration for the example in TCP-AO Test Vectors [I-D.touch-tcpm-ao-test-vectors], Section 5.1.1.

[note: '\' line wrapping for formatting only]

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
 This example sets TCP-AO configuration parameters as
 demonstrated by examples in draft-touch-tcpm-ao-test-vectors.
-->
<config xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <key-chains
      xmlns="urn:ietf:params:xml:ns:yang:ietf-key-chain">
    <key-chain>
      <name>ao-config</name>
      <description>"An example for TCP-AO configuration."</descriptio\
n>
      <key>
        <key-id>61</key-id>
        <crypto-algorithm>hmac-sha-1-12</crypto-algorithm>
        <key-string>
          <hexadecimal-string>01:23:a5:93:b9:db:70:62:9b:be:2c:a6:77:cd:fd:e\
a:6f:e0:ac:ad</hexadecimal-string>
        </key-string>
      </key>
    </key-chain>
  </key-chains>
  <tcp
      xmlns="urn:ietf:params:xml:ns:yang:ietf-tcp">
    <!--
        The example in draft-touch-tcpm-ao-test-vectors uses
        IP addresses that are not valid for examples. Changing
        them here to valid addresses.
    -->
    <server>
      <local-address>192.168.1.1</local-address>
    </server>
    <client>
      <remote-address>192.168.1.2</remote-address>
    </client>
  </tcp>
</config>
```

Appendix D. Complete Tree Diagram

Here is the complete tree diagram for the TCP YANG model.

```
   module: ietf-tcp
     +--rw tcp!
        +--rw connections
        |  +--rw connection*
        |          [local-address remote-address local-port remote-port]
        |     +--rw local-address     inet:ip-address
        |     +--rw remote-address    inet:ip-address
        |     +--rw local-port        inet:port-number
        |     +--rw remote-port       inet:port-number
        |     +--rw common
        |        +--rw keepalives!
        |        |  +--rw idle-time         uint16
        |        |  +--rw max-probes        uint16
        |        |  +--rw probe-interval    uint16
        |        +--rw (authentication)?
        |           +--:(ao)
        |           |  +--rw enable-ao?             boolean
        |           |  +--rw send-id?               uint8
        |           |  +--rw recv-id?               uint8
        |           |  +--rw include-tcp-options?   boolean
        |           |  +--rw accept-key-mismatch?   boolean
        |           +--:(md5)
        |              +--rw enable-md5?            boolean
        +--rw server {server}?
        |  +--rw local-address    inet:ip-address
        |  +--rw local-port?      inet:port-number
        |  +--rw keepalives!
        |     +--rw idle-time         uint16
        |     +--rw max-probes        uint16
        |     +--rw probe-interval    uint16
        +--rw client {client}?
        |  +--rw remote-address    inet:host
        |  +--rw remote-port?      inet:port-number
        |  +--rw local-address?    inet:ip-address
        |  +--rw local-port?       inet:port-number
        |  +--rw keepalives!
        |     +--rw idle-time         uint16
        |     +--rw max-probes        uint16
        |     +--rw probe-interval    uint16
        +--ro statistics {statistics}?
           +--ro active-opens?            yang:counter32
           +--ro passive-opens?           yang:counter32
           +--ro attempt-fails?           yang:counter32
           +--ro establish-resets?        yang:counter32
           +--ro currently-established?    yang:gauge32
           +--ro in-segments?              yang:counter64
           +--ro out-segments?             yang:counter64
           +--ro retransmitted-segments?   yang:counter32
           +--ro in-errors?                yang:counter32
           +--ro out-resets?               yang:counter32
           +---x reset
              +---w input
              |  +---w reset-at?   yang:date-and-time
              +--ro output
                 +--ro reset-finished-at?   yang:date-and-time
```

Authors' Addresses

Michael Scharf
Hochschule Esslingen - University of Applied Sciences
Flandernstr. 101
Esslingen 73732
Germany

Email: michael.scharf@hs-esslingen.de

Vishal Murgai
Samsung

Email: vmurgai@gmail.com

Mahesh Jethanandani
Kloud Services

Email: mjethanandani@gmail.com

TCP Maintenance and Minor Extensions
Internet-Draft
Intended status: Informational
Expires: 6 May 2021

J. Kang, Ed.
Q. Liang
Huawei

2 November 2020

Accurate Data Scheduling by Server in MPTCP

draft-kang-tcpm-accurate-data-scheduling-by-server-01

Abstract

This document defines a new mechanism that enables MPTCP server to send request to MPTCP client for data scheduling between specified subflows during a MPTCP session.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

MPTCP protocol is now being deployed in more and more networks. In most scenarios, MPTCP scheduling strategies for these subflows are implemented on client considering RTT and congestion, or sending packets redundantly. MPTCP server cannot participate in such decision-making.

However, in actual deployment, MPTCP server is configured with multiple network interfaces from different operators. There are some scenarios that MPTCP server wants to set scheduling on these network interfaces to MPTCP client based on its own rules, network planning and operating policies during a MPTCP session. Requirements for these use cases are listed below:

* Network fault prevention. Server tools can help detect quality of deployed network interfaces such as packet loss, delay and jitter. If key performance indicators (KPI) of one network interface becomes worse, MPTCP server hopes to indicate MPTCP client to switch the traffic into another network interface with better KPI.

* A new entrance is deployed or an emergency entrance of 5G card is added during a MPTCP session. In this case, MPTCP server may hope to lead the traffic on some subflows to the additional network interface. One purpose is to test new network in trial operation. Other purposes include avoiding congestion and improving transmission reliability on existing networks. For example, MPTCP server is using network of Operator A for data transmission and a network of operator B is added when data traffic increases. At this time, MPTCP server wants to arrange traffic from network of Operator A to Operator B. In this case, network of Operator B is the target network.

* Value-added services. This requirement comes from the operating policies. MPTCP server is required to help provide customers with diversified services in order to satisfy individual demand. For example,it should be possible that MPTCP server indicates MPTCP client to switch the traffic of VIP users to a network interface with better KPIs.

* Another typical scenario is that MPTCP server hopes to adjust traffic to a specific subflow because of the changes in network cost, for example, the expiration of discounts. This requirement also comes from the operating policies.

There are two related implementations. RFC8684 defines REMOVE_ADDR Option to delete one address during a MPTCP session and it also closes all subflows bound to this address. draft-hoang-mptcp-sub-rate-limit-00 proposes a Subflow Rate Limit Option which can be used by sender to receiver for setting the rate of one subflow to zero. But for the use cases in this document, existing technologies are somewhat inadequate because they do not provide a clear indication of which subflow to switch to.

2. Typical flows for accurate data scheduling by server

This document proposes an accurate data scheduling mechanism for server. Two typical flows are illustrated in Figure 1 and Figure 2.

```
          +--------+                            +--------+
          | Client |                            | Server |
          +--------+                            +--------+
              |                                      |
              |<----Session setup with subflows----->|
              |                                      |
              |                            Determine Address ID
              |                       for target network Interface
              |                                      |
              |<--Sending MP_Navigation for request--|
              |        on one ongoing subflow        |
              |                                      |
      Determine target subflow                       |
     by Address ID in MP_Navigation                  |
              |                                      |
      Traffic switching to                           |
      the target subflow                             |
              |                                      |
              |----Data transfer over the target---->|
              | subflow of target network interface  |
              |                                      |
              |                                      |
```

Figure 1: Server request client to perform traffic switching

```
         +--------+                            +--------+
         | Client |                            | Server |
         +--------+                            +--------+
             |                                      |
             |<---Subflow with diverted traffic --->|
             |       is still kept alive            |
             |                                      |
             |                         Determine to cancel MP_Navigation
             |                            for target network Interface
             |                                      |
             |<----Sending MP_Navigation on the-----|
             |    subflow with diverted traffic     |
             |          for cancellation            |
             |                                      |
  Cancel traffic switching to                       |
   target network interface                         |
     on the MP_Navigation                           |
             |                                      |
             |-----Continue data transfer over----->|
             |   the subflow with diverted traffic  |
             |                                      |
             |                                      |
```

Figure 2: Server sends a request to client to cancel previous navigation setting

For the use case of adding a new network interface to MPTCP session for navigation, normal process of ADD_ADDR should be executed before traffic switching.

If it is determined to cancel the data switching on the subflow, the client should delete the navigation information. The navigation information is generated by the client and is used to determine the target subflow for data switching based on the address ID of the target network interface.

After data switching, if the subflow with diverted traffic is disconnected, the client should delete the navigation information and configuration information for it. The navigation information is generated by the client and is used to determine the target subflow for data switching based on the address ID of the target network interface.

3. Examples

3.1. Traffic switching to a newly-added network interface

Four subflows have been established between client and server that are <IP1, IP3>, <IP2, IP3>, <IP1, IP4> and <IP2, IP4>. On the client, IP1 and IP2 are the address IDs for WiFi and a cellular network. On the server, IP3 and IP4 are the address IDs for Ethernet and WiFi. When a new 5G network is deployed on the server, the server can switch the data traffic on the subflow <IP2,IP4> to the destination IP5 corresponding to 5G. In this case, the target network interface is IP5.

3.2. Traffic switching to a network interface already in the session

Four subflows have been established between client and server that are <IP1, IP3>, <IP2, IP3>, <IP1, IP4> and <IP2, IP4>. On the client, IP1 and IP2 are the address IDs for WiFi and a cellular network. On the server, IP3 and IP4 are the address IDs for Ethernet and WiFi. Server tool detects that KPI for IP4 is better now so the server can switch data traffic on the subflow <IP1, IP3> to the destination IP4.

4. MP_Navigation Option

In this solution, a MP_Navigation option is defined and sent from server to forces client to switch traffic from the subflow over which the option was received to a target subflow or to cancel the traffic switching when it is not required, which is indicated by a flag ’R’. If it is set, the target subflow is determined through the Address ID of the target network interface in MP_Navigation option.

This MP_Navigation Option can be sent in ACK.

It is noted that if this option is not supported by the client, it should be omitted.

4.1. Option Format

The format of the MP_Navigation option is depicted in Figure 3:

```
                     1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+-------+-------+---------------+
|     Kind      |    Length     |Subtype|r|R|E|B|   Address ID  |
+---------------+---------------+-------+-------+---------------+
```

Figure 3: MP_Navigation Option

Subtype: A new subtype should be allocated to indicate MP_Navigation Option.

Flag ’r’: reserved for future usage.

Flag ’R’: when set, defines the content of this option, as follows:

- When value of ’R’ is set to zero, server want to use this option to inform client of navigation policy and the client will perform traffic switching as requested by the server. When value of ’R’ is set to 1, the server requests to cancel current navigation policy in client and the client will delete corresponding navigation information to the Address ID.

Flag ’E’: exists to provide reliability for this option (like that in "ADD_ADDR").

Flag ’B’: indicates whether the subflow over which the option is received is a backup one (that is compatiable with the value by MP_PRIO).

Address ID: Address ID in MP_Navigation Option is used to identify the address ID of target network Interface.

When the client receives the MP_Navigation Option, it will determine the target network interface by the Address ID. Address ID may map to one or more ongoing subflows and the client will select one for each data transfer by its local strategies.

5. Data Scheduling on client when receiving MP_Navigation

This section will be finished later.

6. IANA Considerations

IANA is requested to assign a MPTCP option subtype for the MP_Navigation option.

7. Security Considerations

Since MP_Navigation Option is neither encrypted nor authenticated, on-path attackers and middleboxes could remove, add or modify the MP_Navigation Option on observed Multipath TCP connections.

8. References

8.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.

[RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <https://www.rfc-editor.org/info/rfc8684>.

Authors' Addresses

Jiao Kang (editor)
Huawei
D2-03,Huawei Industrial Base
Shenzhen
China

Phone: +86 18520828326
Email: kangjiao@huawei.com

Qiandeng Liang
Huawei
No. 207, Jiufeng 3rd Road, East Lake High-tech Development Zone
Wuhan
China

Phone: +86 18651640216
Email: liangqiandeng@huawei.com

TCP Maintenance and Minor Extensions
Internet-Draft
Intended status: Informational
Expires: 10 February 2021

J. Kang, Ed.
Q. Liang
Huawei

9 August 2020

Fault Management Mechanism in MPTCP Session
draft-kang-tcpm-fault-management-in-mptcp-session-00

Abstract

This document presents a mechanism for fault management during a MPTCP session. It is used to convey subflow failure information from client to server by other subflow running normally. It includes: 1) a new Fault Announce Option for describing subflow failure, 2) implementation and interoperability of this option during a MPTCP session when one subflow suffers a failure. In fact, the server is able to determine network problems accurately based on these fault information reported from multiple clients for their connections.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

During data transmission in a MPTCP session, subflows may encounter some problems, for example, port failure on one endpoint, network failure, or middlebox working abnormally. Current MPTCP protocol does not provide exchanges between client and server when a fault happens on a subflow which will cause transmission failure or delay.

[RFC8684] introduces TCP RST Reason (MP_TCPRST) option to signal reasons for sending a RST on a subflow which can help an implementation decide whether to attempt later reconnection. TCP RST Reason (MP_TCPRST) option only reports the reason for a specific subflow that has been determined to be closed later. This solution does not cover the case of abnormal termination of one ongoing subflow.

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. Fault Announce Exchanges

This document proposes a fault announce mechanism with a new option that can be used to deliver failure information of abnormal subflow between client and server via another subflow in the MPTCP session that works properly. The flow is illustrated in Figure 1.

```
          +--------+                               +--------+
          | Client |                               | Server |
          +--------+                               +--------+
              |                                         |
              |<---MPTCP Session setup with subflows--->|
              |                                         |
   Determine that one ongoing subflow                   |
         is faulty                                      |
              |                                         |
              |-------Send Fault Announce option------->|
              |     indicating suflow failure via       |
              |           another subflow               |
              |                                         |
              |                                         |
              |                                         |
```

Figure 1: Client sends Fault Announce to server during a MPTCP Session

The Fault Announce option is carried on SYN, ACK or data packets.

Client may detect a local fault, for example, local port or network card failure, or an error in local protocol processing. In this way, the client can determine the fault cause.

Client may actively detect subflow failure by a detecting task to determine the fault cause. For example, the client may deploy a detection task using a bidirectional forwarding detection (BFD) to determine whether the subflow is faulty.

Client may send an ICMP request to server and determine the exceptions by the duration of a response. Specifically, if the client cannot receive a response within a preset time, it means that this subflow is not working properly.

Another way for client to determine the fault reason is ICMP error report. Client may receive an ICMP error report from a third device (e.g., middlebox on the faulty subflow), in which indicates the fault cause.

3. Fault Announce option

A new Fault Announce option is defined to describe the fault in detail occurring on one subflow. If it is set, the faulty subflow is identified by its source address ID (SrcAddressID) and destination address ID (SrcAddressID). The mapping between IP addresses and addresses IDs should be created on both client and server through the process of ADD_ADDR defined in [RFC8684] and [RFC6824].

3.1. Option format

The format of the Fault Announce option (FAULT_ANNOUNCE) is depicted in Figure 2:

```
                      1                   2                   3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
 +---------------+---------------+-------+-------+---------------+
 |     Kind      |    Length     |Subtype| (rsv) |     Cause     |
 +---------------+---------------+-------+-------+---------------+
 | DestAddressID |  SrcAddressID |                               |
 +---------------+---------------+-------------------------------+
```

Figure 2: Fault Announce (FAULT_ANNOUNCE) Option

A new subtype should be allocated to indicate Fault Announce option.

"Cause" is an 8-bit field to describe the reason code for which causes the subflow to malfunction. Client detects the fault and determines the cause. Following values (partially mapped to the Exception Code in ICMP error report) are defined in this document:

* 0x00~0x09 is reserved. It is compatible with "Reason" defined in RFC8684.

* Network is unreachable (code 0x0A).

* Host is unreachable (code 0x0B).

* Routing is failed (code 0x0C).

* Server Suppression (code 0x0D).

* TTL equals zero (IP loops may occur) (code 0x0E).

"SrcAddressID" is used to identify source address ID for the faulty subflow.

"DestAddressID" is used to identify destination address ID for the faulty subflow.

3.2. Additional requirements to be considered

3.2.1. Scenario of middlebox failure

In some actual scenarios, it is the middlebox failure that causes blocking of one subflow. So client should report to server the information of the faulty middlebox by Fault Announce option so that the server can quickly locate it. The information of a faulty middlebox may include:

Middlebox IP: The IP address of the faulty middlebox.

IP protocol version: The IP protocol version adopted by the faulty middlebox, i.e. IPv4 or IPv6. Server can use it to parse the field of "Middlebox IP address".

Flag 'A': If "Middlebox IP address" is optional, this flag should be defined to indicate whether the field of "Middlebox IP address" is carried in Fault Announce option.

3.2.2. Scenario of distinguishing fault types

In some possible implementations, faults are classified into transient fault and non-transitory fault. So a field of "fault type" may be added to identify the type (transient fault or non-transitory fault) for subsequent processing.

4. IANA Considerations

IANA is requested to assign a MPTCP option subtype for the Fault Announce option.

5. Security Considerations

Fault Announce option is neither encrypted nor authenticated, so on-path attackers and middleboxes could remove, add or modify this option on observed Multipath TCP connections.

6. References

6.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.

[RFC8684] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 8684, DOI 10.17487/RFC8684, March 2020, <https://www.rfc-editor.org/info/rfc8684>.

Authors' Addresses

Jiao Kang (editor)
Huawei
D2-03,Huawei Industrial Base
Shenzhen
China

Phone: +86 18520828326
Email: kangjiao@huawei.com

Qiandeng Liang
Huawei
No. 207, Jiufeng 3rd Road, East Lake High-tech Development Zone
Wuhan
China

Phone: +86 18651640216
Email: liangqiandeng@huawei.com

TCP Maintenance Working Group
Internet-Draft
Obsoletes: 6937 (if approved)
Intended status: Standards Track
Expires: 4 May 2021

M. Mathis
N. Dukkipati
Y. Cheng
Google, Inc.

31 October 2020

Proportional Rate Reduction for TCP
draft-mathis-tcpm-rfc6937bis-00

Abstract

This document updates the Proportional Rate Reduction (PRR) algorithm described as experimental in RFC 6937 to standards track. PRR potentially replaces the Fast Recovery and Rate-Halving algorithms. All of these algorithms regulate the amount of data sent by TCP or other transport protocol during loss recovery. PRR more accurately regulates the actual flight size through recovery such that at the end of recovery it will be as close as possible to the ssthresh, as determined by the congestion control algorithm.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document updates the Proportional Rate Reduction (PRR) algorithm described in [RFC6937]from experimental to standards track. PRR accuracy regulates the amount of data sent during loss recovery, such that at the end of recovery the flight size will be as close as possible to the ssthresh, as determined by the congestion control algorithm. PRR has been deployed in at least 3 major operating systems covering the vast majority of today's web traffic. There have been no changes to PRR as documented in the experimental RFC. The descriptions here have been [will be] updated to normative standards language. For a tutorial description of the algorithms and the rationale behind them please see the original RFC.

The experimental RFC describes two different reduction bound algorithms to limit the total window reduction due to all mechanisms, including transient application stalls and the losses themselves: 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 1 segment per ACK. [RFC6937] left the choice of Reduction Bound to the discretion of the implementer.

The paper "An Internet-Wide Analysis of Traffic Policing" [Flatch et al] uncovered a crucial situation where the Reduction Bound mattered. Under certain configurations, token bucket traffic policers [token_bucket] can suddenly start discarding a large fraction of the traffic. This happens without warning when policers run out of tokens. The transport congestion control has no opportunity to measure the token rate, and sets ssthresh based on the recently observed path performance. This value for ssthresh may be substantially larger than can be sustained by the token rate, potentially causing persistent high loss. Under these conditions, both reduction bounds perform very poorly. PRR-CRB is too timid, sometimes causing very long recovery times at smaller than necessary windows, and PRR-SSRB is too aggressive, often causing many retransmissions to be lost multiple times.

Investigating these environments led to the development of a heuristic to dynamically switch between Reduction Bounds: use PRR-SSRB only while snd.una is advancing without additional losses and use PRR-CRB otherwise.

This heuristic is only invoked for what should be a rare corner case: when losses or other events cause the flight size to fall below ssthresh. The extreme loss rates that make the heuristic important are only common in the presence of poorly configured token bucket policers, which are pathologically wasteful and inefficient. In these environments the heuristics serves to salvage a bad situation and any reasonable implementation of the heuristic performs far better than either bound by itself.

The algorithm below is identical to the algorithm presented in [RFC6937]. The "conservative" parameter MAY be replaced by the heuristic also described below.

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]

[All text below is copied from RFC 6937, it will be revised after this document is adopted as a tcpm work item]

2. Definitions

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

RFC 793: snd.una (send unacknowledged)

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 1 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
prr_delivered = 0         // Total bytes delivered during recovery
prr_out = 0               // Total bytes sent during recovery
RecoverFS = snd.nxt-snd.una // FlightSize at the start of recovery
```

Figure 1

```
On every ACK during recovery compute:
   DeliveredData = change_in(snd.una) + change_in(SACKd)
   prr_delivered += DeliveredData
   pipe = (RFC 6675 pipe algorithm)
   if (pipe > ssthresh) {
      // Proportional Rate Reduction
      sndcnt = CEIL(prr_delivered * ssthresh / RecoverFS) - prr_out
   } else {
      // Two versions of the Reduction Bound
      if (conservative) {    // PRR-CRB
        limit = prr_delivered - prr_out
      } else {               // PRR-SSRB
        limit = MAX(prr_delivered - prr_out, DeliveredData) + MSS
      }
      // Attempt to catch up, as permitted by limit
      sndcnt = MIN(ssthresh - pipe, limit)
   }
```

Figure 2

```
On any data transmission or retransmission:
  prr_out += (data sent) // strictly less than or equal to sndcnt
```

Figure 3

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 Additive Increase Multiplicative Decrease (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 2 new segments followed by 1 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 are 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  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.)
```

Figure 4

Note that all 3 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
```

Figure 5

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 5 ACKs trigger exactly 1 transmission each (2 new data, 3 old data), and cwnd is set to 5. At a window size of 5, it takes 3 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 slow start cwnd from 5 up to 10. In this example, TCP operates at half of the window chosen by the congestion control for more than 3 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 is 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 slow start 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 FlightSize and ssthresh, PRR-CRB and PRR-SSRB have identical behaviors.

4. Properties

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

PRR maintains TCP's ACK clocking across most recovery events, including burst losses. RFC 6675 can send large unclocked bursts following burst losses.

Normally, PRR 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, PRR will converge to exactly the target window chosen by the congestion control algorithm. Note that as TCP approaches the end of recovery, prr_delivered will approach RecoverFS and sndcnt will be computed such that prr_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.

PRR 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 (receiver window). When there is an application stall early during recovery, prr_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 prr_delivered - prr_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.

PRR 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, a step discontinuity in the pipe estimator causes a burst of data, which cannot be retracted once the pipe estimator is corrected a few ACKs later. For PRR, pipe merely determines which algorithm, PRR or the

Reduction Bound, is used to compute sndcnt from DeliveredData. While pipe is underestimated, the algorithms are different by at most 1 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/-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 Conservation 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 bursts of data when there are bursts of losses. PRR-SSRB is a compromise that permits TCP to send 1 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 RFC 6675 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 (Section 3.1 in [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 was triggered, thus the various PRR algorithms started in the Reduction Bound phase, and RFC 3517 sent 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 Conservation 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 permits 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 Conservation 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 1 extra segment per ACK relative to the Strong Packet Conservation Bound, to partially compensate for excess losses.

From the perspective of the Strong Packet Conservation 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 PRR and every variant of the Reduction Bound. However, we do not anticipate any future exploration of the alternative Reduction Bounds.

7. Acknowledgements

This document is based in part on previous incomplete work by Matt Mathis, Jeff Semke, and Jamshid Mahdavi [RHID] and influenced by several discussions 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.

Neal Cardwell for reviewing and testing the patch.

8. Security Considerations

PRR 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. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/info/rfc793>.

[RFC2018] Mathis, M., Mahdavi, J., Floyd, S., and A. Romanow, "TCP Selective Acknowledgment Options", RFC 2018, DOI 10.17487/RFC2018, October 1996, <https://www.rfc-editor.org/info/rfc2018>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, DOI 10.17487/RFC5681, September 2009, <https://www.rfc-editor.org/info/rfc5681>.

[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, DOI 10.17487/RFC6675, August 2012, <https://www.rfc-editor.org/info/rfc6675>.

10. Informative References

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

[FACK] Mathis, M. and J. Mahdavi, "Forward Acknowledgment: Refining TCP Congestion Control", ACM SIGCOMM SIGCOMM96, August 1996.

[IMC11] Dukkipati, N., Mathis, M., Cheng, Y., and M. Ghobadi, "Proportional Rate Reduction for TCP", Proceedings of the 11th ACM SIGCOMM Conference on Internet Measurement 2011, Berlin, Germany, November 2011.

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

[Laminar] Mathis, M., "Laminar TCP and the case for refactoring TCP congestion control", Work in Progress, 16 July 2012.

[RFC3042] Allman, M., Balakrishnan, H., and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit", RFC 3042, DOI 10.17487/RFC3042, January 2001, <https://www.rfc-editor.org/info/rfc3042>.

[RFC3517] Blanton, E., Allman, M., Fall, K., and L. Wang, "A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP", RFC 3517, DOI 10.17487/RFC3517, April 2003, <https://www.rfc-editor.org/info/rfc3517>.

[RFC6937] Mathis, M., Dukkipati, N., and Y. Cheng, "Proportional Rate Reduction for TCP", RFC 6937, DOI 10.17487/RFC6937, May 2013, <https://www.rfc-editor.org/info/rfc6937>.

[RHID] Mathis, M., Semke, J., and J. Mahdavi, "The Rate-Halving Algorithm for TCP Congestion Control", Work in Progress, August 1999.

[RHweb] Mathis, M. and J. Mahdavi, "TCP Rate-Halving with Bounding Parameters", Web publication, December 1997, <http://www.psc.edu/networking/papers/FACKnotes/current/>.

[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.

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/-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 that 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 that 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.) is likely to increase the fine grained fluctuations in queue size but does 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 Amphitheatre Parkway
Mountain View, California 94043
United States of America

Email: mattmathis@google.com

Nandita Dukkipati
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, California 94043
United States of America

Email: nanditad@google.com

Yuchung Cheng
Google, Inc.
1600 Amphitheatre Parkway
Mountain View, California 94043
United States of America

Email: ycheng@google.com

TCP Maintenance Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 6, 2021

K. Yang
N. Cardwell
Y. Cheng
E. Dumazet
Google, Inc

November 2, 2020

TCP ETS: Extensible Timestamp Options
draft-yang-tcpm-ets-00

Abstract

This document presents ETS: an Extensible TimeStamps option for TCP. It allows hosts to use microseconds as the unit for timestamps to improve the precision of timestamps, and advertise the maximum ACK delay for its own delayed ACK mechanism. Furthermore, it extends the information provided in the [RFC7323] TCP Timestamps Option by including the receiver delay in the TSecr echoing, so that the receiver of the ACK is able to more accurately estimate the portion of the RTT that resulted from time traveling through the network. The ETS option format is extensible, so that future extensions can add further information without the overhead of extra TCP option kind and length fields.

Status of This Memo

Copyright Notice

1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here. In this document, these words will appear with that interpretation only when in UPPER CASE. Lower case uses of these words are not to be interpreted as carrying [RFC2119] significance.

2. Introduction

Accurate round-trip time (RTT) estimation is necessary for TCP to adapt to diverse and dynamic traffic conditions.

The TCP timestamp option specified in [RFC7323] is designed largely for RTT samples intended for computing TCP's retransmission (RTO) timer [RFC6298].

Some congestion control algorithms may wish to use a form of RTT measurement as one of several congestion signals, since elevated RTT measurements can reflect increases in network queueing delays. For example, the Swift congestion control algorithm [KDJWWM20], successfully deployed in data-center environments, requires precise and accurate measurements of both network and host delays. However, the existing TCP RTT sampling mechanisms that measure the delay between data transmission and ACK receipt [RFC6298] do not separate network and host delays, and cannot measure the RTT of retransmitted data. Even the TCP timestamp option specified in [RFC7323] is not well-suited to use as a congestion signal, for a number of reasons.

With the TCP Timestamps Option [RFC7323], data senders can measure an RTT sample by computing the difference between the data sender's current timestamp clock value and the received TSecr value. However, there are some drawbacks in this [RFC7323] measurement method:

1. The TCP endpoint can compute an [RFC7323] RTT measurement only if the ACK advances the left edge of the send window, i.e. SND.UNA is increased. (Without this rule, in one-way data flows the RTT measured by the receiver would be inflated by gaps in the data sending process; see [RFC7323] Section 4.1).

2. Even if an ACK advances the left edge of the send window, RTT measurements can be greatly inflated by delayed ACKs: the host receiving a data segment can delay an ACK segment in hopes of piggybacking an ACK on the next outgoing data segment. This delay can be as large as 500 milliseconds (as in [RFC1122] Section 4.2.3.2).

3. There is delay included in the RTT measurement that is irrelevant to network queuing, e.g. host-side transmit and receive delays, including delays for waking CPUs from power management "C-states".

4. [RFC7323] specifies a "timestamp clock frequency in the range 1 ms to 1 sec per tick" ([RFC732] Section 5.4). But today's datacenter networks are capable of delivering network packets within tens of microseconds [BMPR17]. In such networks the lower bound of 1 ms limits the usefulness of [RFC7323] timestamps. First, [RFC7323] is incapable of accurate RTT measurements in such networks. Second, [RFC7323] timestamps are not suitable for reverting spurious loss recovery and congestion control responses [RFC4015] due to packet reordering in such networks.

In many of the cases above, an RTT sample computed using [RFC7323] can be inflated for reasons other than network queuing. It is difficult for the ACK receiver to infer how long the non-network delay was, which makes it hard to use an [RFC7323] RTT measurement as a clean signal for congestion control.

Delayed ACKs, as mentioned above, are particularly problematic. TCP receivers typically implement a delayed ACK algorithm. To avoid spurious timeouts due to these delayed ACKs, TCP senders can adapt to this delayed ACK behavior by guessing the maximum delayed ACK value of the remote receiver. Historically, many implementations tended to delay ACKs by up to roughly 200ms [WS95], so some implementations have correspondingly used a minimum RTO of 200ms. However, this imposes a latency penalty that is very large compared to RTTs in some of today's datacenter networks. If receivers had the ability to signal their maximum delayed ACK timer delay, this could allow faster timer-based loss recovery, while still avoiding spurious timeouts.

This document presents ETS: an Extensible TimeStamps option for TCP. ETS extends the information provided in the [RFC7323] TCP Timestamps

Option, adding several features. First, ETS allows connections to use microseconds as the unit for timestamps, to improve the precision of timestamps. Second, ETS allows endpoints to advertise the maximum ACK delay for their own delayed ACK mechanism. Third, ETS allows connections to include information about the delay between data receipt and ACK generation, so that the receiver of the ACK is able to more accurately estimate the portion of the RTT that resulted from time that data and ACK segments spent traveling through the network. Fourth, the ETS option format is extensible, so that future extensions can add further information without the overhead of extra TCP option kind and length fields.

2.1. High-level Design

The ETS protocol has two phases: an exchange of ETS options (ETSopt) in the negotiation handshake in <SYN> and <SYN,ACK> segments, and then ETS options included in all following segments.

In the negotiation handshake, the two senders exchange their MaxACKDel: a hint characterizing the maximum amount of a delay that the sender expects to schedule before acknowledging an incoming data segment.

All segments include EcrDel, the delay in the TSecr echoing process that was inserted by the data receiver, helping the receiver of an ACK to estimate the portion of the RTT delay caused by the network.

An example of a handshake exchange is illustrated below:

```
      TCP A (Client)                                      TCP B (Server)
      ______________                                      ______________
      CLOSED                                                      LISTEN

      #1 SYN-SENT     --- <SYN,TSval=X,TSecr=0,
                           EcrDel=0,MaxACKDel=M1> ----------->  SYN-RCVD

                          <SYN,ACK,TSval=Y,TSecr=X, ----------  SYN-RCVD
      #2 ESTABLISHED  <--  EcrDel=E1,MaxACKDel=M2>

      #3 ESTABLISHED  -- <ACK,TSval=Z,TSecr=Y,EcrDel=E2> --> ESTABLISHED
```

Active connect: An actively connecting host that wishes to negotiate ETSopt MUST include the ETSopt in the <SYN>. For backward compatibility, the endpoint performing the active connect MAY also include a [RFC7323] TSopt in the <SYN> segment, so that if the passive side or middleboxes do not support and respond to the ETSopt, the active and passive sides can proceed with the [RFC7323] TSopt negotiation for the connection.

Passive connect: For a passively connecting host that is willing to proceed with ETSopt negotiation, if the <SYN> includes an ETSopt, the host MUST include a TCP ETS option in the initial <SYN,ACK> segment. A retransmission of the <SYN,ACK> segment may omit the ETSopt, to increase robustness in the presence of middleboxes that block segments containing ETSopt.

Processing of <SYN,ACK> for active connect: If the ETSopt is absent from the <SYN,ACK> segment received by the actively connecting endpoint, suggesting that the passive endpoint does not support ETSopt, or some middlebox has stripped the option from the <SYN,ACK> segment, then the actively connecting endpoint MUST disable ETSopt for this connection. In such cases the actively connecting endpoint MAY fall back to using [RFC7323] timestamps if both the <SYN> and <SYN,ACK> segments include valid [RFC7323] timestamps.

3. Detailed Protocol

3.1. Definitions

The reader is expected to be familiar with the TCP Timestamps Option (TSopt), including TSval, TSecr, and TS.Recent [RFC7323].

Variables introduced by this document are described below:

MaxACKDel: a hint characterizing the maximum amount of a delay that the sender expects to schedule before acknowledging an incoming data segment.

TSval: the value of the sender’s timestamp clock when this segment is scheduled for transmission, in microseconds.

TSecr: the echo of TS.recent

TS.Recent: the recently received TSval sent by the remote TCP endpoint in the TSval field of an ETSopt, updated using the TS.Recent rules specified in [RFC7323].

EcrDel: the field quantifying the delay between data receipt and ACK generation, so that the receiver of the ACK is able to more accurately estimate the NetworkRTT.

NetworkRTT: the time from when the data segment leaves the sender until when it arrives at the receiver, plus the time from when the corresponding ACK leaves the (data) receiver until the ACK arrives at the data sender (here sender and receiver refer to the TCP layer only).

3.2. TCP ETS option header format

The header format for TCP ETS options (ETSopt) is as follows:

```
               01234567 89012345 67890123 45678901
              +--------+--------+--------+--------+
              |  Kind  | Length |       ExID      |
              +--------+--------+--------+--------+
              |               TSval               |
              +--------+--------+--------+--------+
              |               TSecr               |
              +--------+--------+--------+--------+
              |Un|   EcrDel   |R|    MaxACKDel    |
              +-----------------+-----------------+
              |                 |
             /                   \
        .---'                     `---------------------.
       /                                                 \
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
      |  Unit  |            EcrDel               |Reserved|
      +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
        2 bits |            13 bits              | 1 bit
```

```
      Kind: 1 byte, has value 254,
            TCP experimental option codepoint [RFC6994]
      Length: 1 byte option length, value is 16 if SYN bit is set,
              otherwise 14 if SYN bit is not set
              (value MAY be higher in later ETSOpt protocol versions).
      ExID: 2 byte [RFC6994] experiment ID; MUST be 0x4554.
      TSval and TSecr: 32 bits each, have the same definition as
                    [RFC7323] except that both are in microseconds.
      EcrDelUnit: 2 bits, has value:
                  0: indicates EcrDel is in microsecond units
                  1: indicates EcrDel is in millisecond units
                  2: indicates EcrDel is invalid
                  3: reserved
      EcrDel: 13 bits, the value of EcrDel.
      Reserved: 1 bit, in this protocol version; sender MUST set to 0;
                receiver MUST ignore field and tolerate 0 or 1
      MaxACKDel: 16 bits, max expected ACK delay in microseconds,
                 MUST only be present when SYN bit is set.
```

The semantics of the option fields are as follows:

TSval: The TSval field contains the value of the sender host's timestamp clock, in microseconds, at the time the sender schedules transmission of the segment.

TSecr: The TSecr field contains the current value of TS.Recent, the recently received TSval sent by the remote TCP that is recorded using the TS.recent update algorithm described in [RFC7323], section 4.3. TSecr is only valid when the ACK bit is set. When the ACK bit is not set, senders MUST set this field to 0, and receivers MUST ignore the value in this field (and MUST tolerate any value in this field).

MaxACKDel: When the SYN bit is set, the sender advertises MaxACKDel as a hint characterizing the maximum amount of a delay that the sender expects to apply before ACKing an incoming data segment. The unit of MaxACKDel is microseconds. Field MaxACKDel MUST be set to 0xFFFE if the value is equal to or greater than 0xFFFE. If the MaxACKDel value is invalid, or the sender does not want to advertise this value, field MaxACKDel MUST be set to the max value allowed by the field, i.e. 0xFFFF. When a host performing an active TCP connection attempt is willing to enable ETS, in a <SYN> segment it MUST include a TCP ETS option (ETSopt) with the MaxACKDel field. If the SYN bit is not set, the field MaxACKDel MUST NOT be present in the option.

EcrDel: Field EcrDel contains the delay inserted by the receiver in this TSecr echoing process. Field EcrDel is only valid when the ACK bit is set, otherwise MUST be set to 0 by the sender and ignored by the receiver. When the ACK bit is set, the sender computes the EcrDel using the following algorithm:

```
(1) When a <SYN> or non-empty data segment SEG is received:
        If SYN is set, or SEG.TSval is after TS.Latest:
            TS.Latest = SEG.TSval
            TS.LatestClock = ArrivalTime of SEG
(2) When an ETSopt is sent:
        TSecr = TS.Recent (as in [RFC7323])
        LatestACKDel = ACKSendTime - TS.LatestClock
        TSecrAge = TS.Latest - TSecr
        EcrDel = LatestACKDel + TSecrAge
```

In (1), a non-empty data segment is defined as a segment containing a non-empty data payload. The timestamps are in a modular 32-bit space, and a timestamp s is "after" timestamp t if and only if $0 < (s - t) < 2^{31}$.

In (2), the semantics of the two temporary variables LatestACKDel and TSecrAge are as follows:

LatestACKDel, the time from the arrival of the segment with the latest TSval until the time when an ACK segment is sent.

TSecrAge, the age of TSecr compared to TS.Latest.

We discuss how the ACK receiver can estimate the latest NetworkRTT using EcrDel in the next section.

3.3. Estimating the Network RTT

NetworkRTT is the time from when the data segment leaves the sender until when it arrives at the receiver, plus the time from when the corresponding ACK leaves the (data) receiver until the ACK arrives at the data sender (here sender and receiver refer to the TCP layer only).

With EcrDel in ETSopt when a data sender (TCP A) receives an ACK segment with ETSopt from the remote endpoint (TCP B), NetworkRTT can be eatimated by:

```
  NetworkRTT = ACKArrivalTime - SEG.TSecr - SEG.EcrDel
```

where ACKArrivalTime is the TCP A’s timestamp clock when the ACK segment arrives, and SEG.TSecr and SEG.EcrDel are the values from the incoming ETSopt.

This gives us the latest network RTT measurement by:

```
NetworkRTT = ACKArrivalTime - SEG.TSecr - SEG.EcrDel
           = ACKArrivalTime - SEG.TSecr - B.TSecrAge - B.LatestACKDel
           = ACKArrivalTime - (SEG.TSecr + B.TSecrAge)
             - B.LatestACKDel
           = ACKArrivalTime - (SEG.TSecr + B.TS.Latest - SEG.TSecr)
             - B.LatestACKDel
           = ACKArrivalTime - B.TS.Latest - B.LatestACKDel
```

In the calculation above, B.TSecrAge, B.TS.Latest and B.LatestACKDel are TSecrAge, TS.Latest and LatestACKDel on TCP B respectively at the time this ACK segment was sent.

At the time TCP B sends an ETSopt, TS.Latest can be different from TS.Recent, e.g. if there is a sequence hole in B’s receiving window. If this is the case, the NetworkRTT measures the network RTT of the latest segment received by TCP B.

The following example shows how NetworkRTT is computed when TS.Latest is different from TS.Recent:

```
      TCP A                                                      TCP B
      ______________                                    ______________
                    -- <TSval=1> -->                       arrives at t=2
                                                  TS.Latest=TS.Recent=1
                                                       TS.LatestClock=2
                    -- <TSval=2> --> lost
                    -- <TSval=3> -->                      arrives at t=10
                                               TS.Recent=1, TS.Latest=3
                                                      TS.LatestClock=10
     arrive at t=11   <-- <ACK, TSecr=1, EcrDel=2> --   Send ACK at t=10
```

In this example, EcrDel is LatestACKDel + TSecrAge = (10 - 10) + (3 - 1) = 2, and the NetworkRTT from the last <ACK> segment is estimated as 11 - 1 - 2 = 8, which is the network RTT of the segment sent with TSval = 3.

In order to make use of this NetworkRTT estimate as a clean signal that more precisely reflects network queuing, it is RECOMMENDED that timestamps ACKArrivalTime and TS.LatestClock use the time at which the segment arrives at the host, e.g. the time the Network Interface Controller (NIC) receives the segment, if the NIC supports hardware receive timestamping. Within this context, NetworkRTT is then defined as the time from when the data segment leaves the sender's TCP until when it is received at the receiver's NIC, plus the time from when the corresponding ACK leaves the (data) receiver's TCP until the ACK is received at the data sender's NIC.

4. Interaction with other TCP Mechanisms

4.1. Interaction with PAWS

Protection Against Wrapped Sequences (PAWS), introduced by [RFC7323] Section 5, is a mechanism to reject old duplicate segments that might corrupt an open TCP connection. In the PAWS mechanism, a segment can be discarded as an old duplicate if it is received with a timestamp SEG.TSval that is "before" some timestamps recently received on this connection.

As in [RFC7323], ETS receivers need to exercise care to avoid spurious PAWS discards due to wrapping 32-bit timestamp values during periods in which the connection is idle. When microsecond units are used, as in ETS, the 32-bit timestamp could trigger wrapping issues and spurious PAWS discards after 2^31 ticks of idleness, which is around 2147 seconds (or around 35.7 minutes). To prevent a false positive PAWS rejection of a valid segment, an ETS receiver MUST skip the PAWS check for the first arriving segment after the timestamp used by PAWS, e.g. TS.Recent, has not been updated for 2147 seconds or more.

4.2. Eifel Considerations

The Eifel Detection Algorithm [RFC3522] detects a spurious recovery by comparing a received TSecr to RetransmitTS, the value of the TSval in the retransmit sent when loss recovery is initiated. ETS allows Eifel to work as-is because the fields TSval and TSecr in the ETSopt have the same semantics as in TSopt [RFC7323]. Further in sub-millisecond environment, ETS microsecond precision is more effective detecting spurious retransmission compared to TSopt's more coarse unit.

4.3. RTO Calculation Considerations

The RTT measurement used in the calculation of RTO (retransmission timeout) [RFC6298] stays the same as described in [RFC7323]. It is NOT RECOMMENDED to use only NetworkRTT measurements for RTO calculation because RTO needs to include the host side delays to avoid spurious RTO events due to host delays.

With MaxACKDel information exchanged in the ETSopt of <SYN> and <SYN,ACK>, senders MAY impose a minimum RTO that is greater than or equal to the MaxACKDel.

4.4. SACK Considerations

The ETSopt has a length of 14 bytes on non-<SYN> segments. This leaves a remaining space of 26 bytes for other TCP options. A SACK option [RFC2018] with at most 3 SACK blocks is able to coexist with ETSopt in a single TCP segment as with TSopt.

4.5. Interaction with Middleboxes

[HNRGHT11] shows that middleboxes could drop an unrecognized TCP option or even drop the whole segment.

In order to fall back on [RFC7323] TSopt, the sender MAY include a [RFC7323] TSopt in the <SYN> and <SYN,ACK> segments, so that the [RFC7323] TSopt can be adopted when the ETSopt is stripped by a middlebox.

Once an expected ETSopt is missing from an incoming segment, the sender MUST NOT include an ETSopt for all future segments of this TCP connection. An implementation could negatively cache such incidents to avoid using ETS on these hosts or routes on future connections.

5. IANA Considerations

This document specifies a new TCP option that uses the shared experimental options format [RFC6994], with ExID in network-standard byte order.

The authors plan to request the allocation of ExID value 0x4554 for the TCP option specified in this document.

6. Security Considerations

A malicious receiver can manipulate the sender's network RTT estimated by forging an EcrDel value. However this does not introduce a new vulnerability relative to the Timestamp option [RFC7323], because a malicious receiver could already forge the TSecr field to manipulate the RTT measured by the other side.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[RFC6994] Touch, J., "Shared Use of Experimental TCP Options", August 2013.

7.2. Informative References

[BMPR17] Barroso, L., Marty, M., Patterson, D., and P. Ranganathan, "Attack of the Killer Microseconds", Communications of the ACM , April 2017.

[HNRGHT11]
Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., and H. Tokuda, "Is it still possible to extend TCP?", InProceedings of the 2011 ACM SIGCOMM conference on Internet measurement conference 2011 Nov 2 (pp.181-194) , 2011.

[KDJWWM20] Kumar, G., Dukkipati, N., Jang, K., Wassel, HM., Wu, X., Montazeri, B., Wang, Y., Springborn, K., Alfeld, C., Ryan, M., and D. Wetherall, "Swift: Delay is Simple and Effective for Congestion Control in the Datacenter", InProceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication 2020 Jul 30 (pp514-528) , 2020.

[RFC1122] Braden, R., "Requirements for Internet hosts-communication layers", October 1989.

[RFC3522] Ludwig, R. and M. Meyer, "The Eifel detection algorithm for TCP", April 2003.

[RFC4015] Ludwig, R. and A. Gurtov, "The Eifel response algorithm for TCP", February 2005.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", June 2011.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, "TCP Extensions for High Performance", September 2014.

[WS95] Wright, G. and W. Stevens, "TCP/IP Illustrated, Volume 2: The Implementation", 1995.

Authors' Addresses

Kevin (Yudong) Yang
Google, Inc

Email: yyd@google.com

Neal Cardwell
Google, Inc

Email: ncardwell@google.com

Yuchung Cheng
Google, Inc

Email: ycheng@google.com

Eric Dumazet
Google, Inc

Email: edumazet@google.com