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 is submitted in full conformance with the provisions of BCP 78 and BCP 79.

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

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

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction .............................. 3
  1.1. Terminology ............................ 7
2. Implementation without MPTCP protocol adaptation .......... 8
   2.1. Re-transmission Timer(RobE_TIMER) ........ 8
   2.2. Simultaneous Initial Paths Simple Version (RobE_SIM) .. 9
   2.3. Heuristic Initial Path Selection (RobE_IPS) .......... 10
      2.3.1. Architecture .......................... 10
      2.3.2. Typical Scenarios ..................... 11
      2.3.3. Path decision information ............. 14
      2.3.4. Initial Path Selection use local RTT information .. 15
   2.4. Combination of RobE_SIM and RobE_IPS ............. 15
   2.5. Combination of RobE_TIMER and RobE_IPS ........... 16
3. Implementation with Bi-directional MPTCP Support .......... 17
   3.1. Simultaneous Initial Paths Extended Version (RobE_eSIM) .................. 18
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 an MP_CAPABLE exchanged first, followed when successfully 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.
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 correspond to different access 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 both a cellular connection and WiFi are present in a mobile phone, WiFi is usually the interface offered to initiate the MPTCP session.

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

Introduction of RobE_SIM and RobE_eSIM aims to overcome the limitations of [RFC6824] and [RFC8684], using one initial flow and introduces the concept of multiple potential initial flows triggered simultaneously. Potential initial flows give 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 offers 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].
<table>
<thead>
<tr>
<th>Scenario</th>
<th>MPTCP</th>
<th>RobE_TIMER</th>
<th>RobE_SIM</th>
<th>RobE_eSIM</th>
<th>RobE_IPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP packet loss</td>
<td>Delayed connection</td>
<td>In the scope of timer</td>
<td>No impact</td>
<td>No impact</td>
<td>Delayed connection</td>
</tr>
<tr>
<td>IP broken</td>
<td>No connection</td>
<td>In the scope of timer</td>
<td>No impact</td>
<td>No impact</td>
<td>No connection</td>
</tr>
<tr>
<td>IP setup duration dependency</td>
<td>Default route</td>
<td>Default route (+ path 1..n)</td>
<td>Fastest path</td>
<td>Fastest path</td>
<td>Selected path</td>
</tr>
<tr>
<td>MP availability duration</td>
<td>MP_CAPABLE HS + MP_JOIN HS</td>
<td>sum_1..n(MP_CAPABLE_n HS) + MP_JOIN HS</td>
<td>MP_CAPABLE max(MP_CAPABLE_1.. MP_CAPABLE_n HS) + MP_JOIN HS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guaranteeing session setup</td>
<td>Depends on the default route</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Depends on selection</td>
</tr>
</tbody>
</table>

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 lack of the full optimization potential which requires protocol adaptation as detailed in Section 3. Following sections will describe the newly introduced mechanisms 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 in the same way as 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 messages might be delayed until the client retries sending them 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.
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 prior negotiation with the receiver side 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.

![Connection Diagram](image.png)

[*] 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.
2.3.2. Typical Scenarios

Two typical RobE_IPS scenarios are presented in this section. Figure 5 shows 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 is 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.

![Figure 4: Architecture for Initial-path Selection](image)
Figure 5: RobE_IPS for each connection establishment
Figure 6 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.
2.3.3. Path decision information

The level of heuristic can be mainly divided into three layers: 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 an 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 an "Initial Path Selection" logic based on RTT, e.g. assuming two paths over LTE and WiFi access. RTT calculation on the transport layer usually reflects the time when an information is sent and a 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 have a limited meaning only when it does not represent the end-to-end latency.

```
+-------------------+
<table>
<thead>
<tr>
<th>New Session</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>Running Connections</td>
</tr>
<tr>
<td>(LTE.RTT&lt;WiFi.RTT)</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>Yes</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Set LTE as initial path</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Set WiFi as initial path</td>
</tr>
</tbody>
</table>
```

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 achieve 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 without any path information followed by RobE_IPS for consecutive connections.
2.5. Combination of RobE_TIMER and RobE_IPS

Since RobE_IPS solely does not guarantee that a session can be set up based 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.
3. Implementation with Bi-directional MPTCP Support

Solutions which require 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 profiting from an explicit support of the remote end host and will be 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 one described in Section 3.1.1 or an explicit one which is described 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 paths 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 proceeds 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 an MP_CAPABLE (SA6.2).

![Figure 11: MPTCP RobE_eSIM implicit Connection Setup](image)

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

<table>
<thead>
<tr>
<th>SYN+MP_CAPABLE+ROBE_eSIM_EN(Key-A[*])</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN+MP_CAPABLE+ROBE_eSIM_EN(Key-A’[*])</td>
</tr>
<tr>
<td>SYN/ACK+MP_CAPABLE+ROBE_eSIM_EN(Key-B)</td>
</tr>
<tr>
<td>SY&lt;ACK+MP_CAPABLE+ROBE_eSIM_EN(Key-B’)</td>
</tr>
<tr>
<td>ACK+MP_CAPABLE(Key-A,Key-B)</td>
</tr>
<tr>
<td>ACK+MP_JOIN_CAP(Key-A, HMAC)</td>
</tr>
</tbody>
</table>
```

[*] 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

```
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
### 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 is 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"]
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.
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.
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 receiving 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.
A successful negotiation allows the exchange of the measured RTT value from one subflow of an 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.
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]

```
<table>
<thead>
<tr>
<th>Host A</th>
<th>Host B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address A1</td>
<td>Address A2</td>
</tr>
<tr>
<td>SYN+MP_CAPABLE+L_RTT_EN</td>
<td>SYN/ACK+MPTCP_CAPABLE</td>
</tr>
<tr>
<td>SYN/ACK+MPTCP_CAPABLE</td>
<td>ACK+MPTCP_CAPABLE</td>
</tr>
</tbody>
</table>
```

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.

```
<table>
<thead>
<tr>
<th>Value</th>
<th>Symbol</th>
<th>Name</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD</td>
<td>ROBE_eSIM_EN</td>
<td>RobE_eSIM enabled</td>
<td>Section 3.1</td>
</tr>
<tr>
<td>TBD</td>
<td>MP_JOIN_CAP</td>
<td>Join connection directly in RobE_eSIM</td>
<td>Section 3.1</td>
</tr>
<tr>
<td>TBD</td>
<td>L_RTT_EN</td>
<td>Server RTT enabled</td>
<td>Section 3.2</td>
</tr>
</tbody>
</table>
```

Table 2: RobE Option Subtypes
5. References

5.1. Normative References


5.2. Informative References


Authors’ Addresses

Markus Amend
Deutsche Telekom
Deutsche-Telekom-Allee 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
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 request the ACK rate to be used by a receiver, and it also allows to request immediate ACKs from a receiver.

Status of This Memo

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

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

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

This Internet-Draft will expire on 2 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document.
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 request 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 (with the appropriate Length value, see Section 4) in packets that have the SYN bit set.

Upon reception of a SYN segment carrying the TARR option, a TARR-option-capable endpoint MUST include the TARR option in the SYN-ACK segment sent in response.

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 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 data segments received from the sender. This request is performed by the sender by including the TARR option in the TCP header of a segment. The TARR option carries the R value requested by the sender (see section 4).

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 1.

A TCP segment carrying retransmitted data is not required to include a TARR option.

A TCP sender that uses time-based loss detection (e.g. RACK-TLP [RFC8985]) MAY use the Ignore Order feature, by which the sender indicates that it has a reordering tolerance of R packets. Otherwise, such feature SHOULD NOT be used. The Ignore Order feature can be used 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 segment.

A TARR-option-capable receiving TCP SHOULD modify its ACK rate to one ACK every R received data segments from the sender. If a TARR-option-capable TCP receives a segment carrying the TARR option with R=1, the receiving TCP SHOULD send an ACK immediately.
If packet reordering occurs, a TCP receiver should send an immediate duplicate ACK when an out-of-order segment arrives [RFC2581], thus in such cases the TCP receiver will not comply with the request. Note also that the receiver might be unable to send ACKs at the requested rate (e.g., due to lack of resources); on the other hand, the receiver might opt not to fulfill a request for security reasons (e.g., to avoid or mitigate an attack by which a large number of senders request disabling delayed ACKs simultaneously and send a large number of data segments to the receiver).

The request to modify the ACK rate of the receiver holds until the next segment carrying a TARR option is received.

A TARR-option-capable TCP that receives a TARR option with the Ignore Order (I) 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.

4. Option Format

The TARR option presents two different formats that can be identified by the corresponding format length. For packets that have the SYN bit set, the TARR option has the format shown in Fig. 1.

```
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      |     Length    |              ExID             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: TCP ACK Rate Request option format for packets that have the SYN bit set.

Kind: The Kind field value is TBD.

Length: The Length field value is 4 bytes.

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

For packets that do not have the SYN bit set, the TARR option has the format and content shown in Fig. 2.
Figure 2: TCP ACK Rate Request option format.

Kind: The Kind field value is TBD.

Length: The Length field value is 5 bytes.

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

R: The size of this field is 6 bits. The field carries the ACK rate requested by the sender. The minimum value of R is 1.

OPTION 1: the R field corresponds to the binary encoding of the requested ACK rate. The maximum value of R is 63. A receiver MUST ignore an R field with all bits set to zero.

OPTION 2: the R field is composed of two subfields: the 4 leftmost bits represent a mantissa (m) and the 2 rightmost bits represent an exponent (e). The value of the requested ACK rate is obtained as R = (m+1)*2^(2*e). The maximum value of R is 1024.

Reserved (V): The size of this field is 1 bit. This bit is reserved for future use.

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

5. Changing the ACK rate during the lifetime of a TCP connection

In some scenarios, especially when conditions are rather stable and/or predictable, setting the ACK rate once for the whole lifetime of a TCP connection may be suitable. However, there are also cases where it may be desirable to modify the ACK rate during the lifetime of a connection.
The ACK rate to be used may depend on the cwnd value used by the
sender, which can change over the lifetime of a connection. cwnd will
start at a low value and grow rapidly during the slow-start phase,
then settle into a reasonably consistent range for the congestion-
avoidance phase - assuming the underlying bandwidth-delay product
(BDP) remains constant. Phenomena such as routing updates, link
capacity changes or path load changes may modify the underlying BDP
significantly; the cwnd should be expected to change accordingly,
prompting the need for ACK rate updates.

TARR can also be used to suppress Delayed ACKs in order to allow
measuring the RTT of each packet in specific intervals (e.g., during
flow start-up), and allow a different ACK rate afterwards.

A Linux receiver has a heuristic to detect slow start and suppress
Delayed ACKs just for that period. However, some slow start variants
(e.g., HyStart, HyStart++, etc.) may alter the ending of slow start,
thus confusing the heuristics of the receiver. To avoid slow start
sender behavior ossification, an explicit signal such as TARR may be
useful.

Another reason to modify the ACK rate might be reducing the ACK load.
The sender may notice that the ACKs it receives cover more segments
than the ACK rate requested, indicating that ACK decimation is
occurring en route. The sender may then decide to reduce the ACK
frequency to reduce receiver workload and network load up to the ACK
decimation point.

Future TCP specifications may also permit Congestion Experienced (CE)
marks to appear on pure ACKs [I-D.ietf-tcpm-generalized-ecn]. This
might involve more frequent ACK rate updates (e.g., once an RTT), as
the sender probes around an operating point.

6. 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.

7. Security Considerations

The TARR option opens the door to new security threats. This section
discusses such new threats, and suggests mitigation techniques.
An attacker might be able to impersonate a legitimate sender, and forge an apparently valid packet intended for the receiver, in order to intentionally communicate a bad \( R \) value to the latter with the aim to damage communication or device performance. For example, in a small \( cwnd \) scenario, using a too high \( R \) value may lead to exacerbated RTT increase and throughput decrease. In other scenarios, a too low \( R \) value may contribute to depleting the energy of a battery-operated receiver at a faster rate or may lead to increased network packet load.

While Transport Layer Security (TLS) [RFC8446] is strongly recommended for securing TCP-based communication, TLS does not protect TCP headers, and thus cannot protect the TARR option fields carried by a segment. One approach to address the problem is using network-layer protection, such as Internet Protocol Security (IPsec) [RFC4301]. Another solution is using the TCP Authentication Option (TCP-AO), which provides TCP segment integrity and protection against replay attacks [RFC5925].

While it is relatively hard for an off-path attacker to attack an unprotected TCP session, it is RECOMMENDED for a TARR receiver to use the guidance and attack mitigation given in [RFC5961]. The TARR option MUST be ignored on a packet that is deemed invalid.

A TARR receiver might opt not to fulfill a request to avoid or mitigate an attack by which a large number of senders request disabling delayed ACKs simultaneously and send a large number of data segments to the receiver.

8. 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. Jonathan Morton and Bob Briscoe provided the main input for Section 5.

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.

9. References

9.1. Normative References
9.2. Informative References


Authors’ Addresses

Carles Gomez
UPC
C/Esteve Terradas, 7
08860 Castelldefels
Spain
Email: carlesgo@entel.upc.edu

Jon Crowcroft
University of Cambridge
JJ Thomson Avenue
Cambridge
United Kingdom
Email: jon.crowcroft@cl.cam.ac.uk
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 was originally 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 updates the original ECN specification to specify 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 previously assigned to the ECN-Nonce. 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. The document also specifies the treatment of this updated TCP wire protocol by middleboxes, updating BCP 69 with respect to ACK filtering.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 8, 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the
document authors. All rights reserved.

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

Table of Contents

1. Introduction ................................................. 3
   1.1. Document Roadmap ....................................... 5
   1.2. Goals ................................................. 5
   1.3. Terminology ........................................... 6
   1.4. Recap of Existing ECN feedback in IP/TCP ............... 6
2. AccECN Protocol Overview and Rationale ........................ 8
   2.1. Capability Negotiation ................................. 9
   2.2. Feedback Mechanism .................................... 9
   2.3. Delayed ACKs and Resilience Against ACK Loss .......... 9
   2.4. Feedback Metrics ...................................... 10
   2.5. Generic (Dumb) Reflector .............................. 11
3. AccECN Protocol Specification ................................. 12
   3.1. Negotiating to use AccECN .............................. 12
       3.1.1. Negotiation during the TCP handshake ............. 12
       3.1.2. Backward Compatibility ........................... 13
       3.1.3. Forward Compatibility ............................ 15
       3.1.4. Retransmission of the SYN ......................... 15
       3.1.5. Implications of AccECN Mode ..................... 16
   3.2. AccECN Feedback ....................................... 18
       3.2.1. Initialization of Feedback Counters .............. 18
       3.2.2. The ACE Field .................................... 19
           3.2.2.1. ACE Field on the ACK of the SYN/ACK ....... 20
           3.2.2.2. Encoding and Decoding Feedback in the ACE Field 21
           3.2.2.3. Testing for Mangling of the IP/ECN Field .... 23
           3.2.2.4. Testing for Zeroing of the ACE Field ........ 24
           3.2.2.5. Safety against Ambiguity of the ACE Field ... 25
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
Internet-Draft  Accurate TCP-ECN Feedback  March 2022

(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 document 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.

<table>
<thead>
<tr>
<th>IP-ECN codepoint</th>
<th>Codepoint name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0b00</td>
<td>Not-ECT</td>
<td>Not ECN-Capable Transport</td>
</tr>
<tr>
<td>0b01</td>
<td>ECT(1)</td>
<td>ECN-Capable Transport (1)</td>
</tr>
<tr>
<td>0b10</td>
<td>ECT(0)</td>
<td>ECN-Capable Transport (0)</td>
</tr>
<tr>
<td>0b11</td>
<td>CE</td>
<td>Congestion Experienced</td>
</tr>
</tbody>
</table>

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.
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 for the Data Receiver to feed back the number of packets arriving with CE in the IP-ECN field. 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 in the IP-ECN field (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 respectively marked CE, ECT(1) and ECT(0) in the IP-ECN field. 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 feedback about CE markings in the IP-ECN field of both data and control packets. According to [RFC3168] the Data Sender is meant to set the IP-ECN field of 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 IP-ECN field. 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 IP-ECN field when it arrives at the receiver could still be useful, because middleboxes have been known to overwrite the IP-ECN 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 IP-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. Previously, if a TCP server received ECT or CE on a SYN, it could not know whether it was invalid (or valid) because only the TCP client knew 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.

```
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               |           | 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 AE, CWR and ECE TCP flags on the SYN/ACK to the combination in the top block of 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).

When the TCP server returns any of the 4 combinations in the top block of Table 2, it confirms 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.

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.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>SYN A→B</th>
<th>SYN/ACK B→A</th>
<th>Feedback Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccECN</td>
<td>AccECN</td>
<td>1 1 1</td>
<td>0 1 0</td>
<td>AccECN (no ECT on SYN)</td>
</tr>
<tr>
<td>AccECN</td>
<td>AccECN</td>
<td>1 1 1</td>
<td>0 1 1</td>
<td>AccECN (ECT1 on SYN)</td>
</tr>
<tr>
<td>AccECN</td>
<td>AccECN</td>
<td>1 1 1</td>
<td>1 0 0</td>
<td>AccECN (ECT0 on SYN)</td>
</tr>
<tr>
<td>AccECN</td>
<td>AccECN</td>
<td>1 1 1</td>
<td>1 1 0</td>
<td>AccECN (CE on SYN)</td>
</tr>
<tr>
<td>AccECN</td>
<td>Nonce</td>
<td>1 1 1</td>
<td>1 0 1</td>
<td>(Reserved)</td>
</tr>
<tr>
<td>AccECN</td>
<td>ECN</td>
<td>1 1 1</td>
<td>0 0 1</td>
<td>classic ECN</td>
</tr>
<tr>
<td>AccECN</td>
<td>No ECN</td>
<td>1 1 1</td>
<td>0 0 0</td>
<td>Not ECN</td>
</tr>
<tr>
<td>Nonce</td>
<td>AccECN</td>
<td>0 1 1</td>
<td>0 0 1</td>
<td>classic ECN</td>
</tr>
<tr>
<td>ECN</td>
<td>AccECN</td>
<td>0 1 1</td>
<td>0 0 1</td>
<td>classic ECN</td>
</tr>
<tr>
<td>No ECN</td>
<td>AccECN</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>Not ECN</td>
</tr>
<tr>
<td>AccECN</td>
<td>Broken</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>Not ECN</td>
</tr>
</tbody>
</table>

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 SYN 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, if one peer were to negotiate the feedback mode in two different types of handshake, it would not be possible for the other peer to know for certain which handshake packet(s) the other end had eventually received or in which order it received them. So, in the absence of these rules, the two peers could end up using different feedback modes without knowing it.

  o Congestion response:
    * It is still obliged to respond appropriately to AccECN feedback that indicates there were ECN marks 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:

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

  o 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.
o 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.

o 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, as 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).

- A Data Receiver that supports sending of the AccECN TCP Option 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 optionally 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 = r.elb = 1 and r.ceb = 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 \( s.cep = 5 \), \( s.e0b = s.elb = 1 \) and \( s.ceb = 0 \).

### 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
+------------------------------------------+
| Header Length | Reserved | ACE | R | A | P | R | 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.

+---------------------+---------------------+-----------------------+
<p>| IP-ECN codepoint on | ACE on pure ACK of   | r.cep of client in    |</p>
<table>
<thead>
<tr>
<th>SYN/ACK</th>
<th>SYN/ACK</th>
<th>AccECN mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Not-ECT</td>
<td>0b010</td>
<td>5</td>
</tr>
<tr>
<td>ECT(1)</td>
<td>0b011</td>
<td>5</td>
</tr>
<tr>
<td>ECT(0)</td>
<td>0b100</td>
<td>5</td>
</tr>
<tr>
<td>CE</td>
<td>0b110</td>
<td>6</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------</td>
<td>-----------------------</td>
</tr>
</tbody>
</table>

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.
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 3.2.2.4).

(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).

- 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).

- 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 using the encoding in Table 2 in the SYN/ACK. 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. A server MAY implement a test for such a case, but it is not required. Therefore, once in the ESTABLISHED state, it will be sufficient for the server to consider the ACE field to be encoded as the normal ACE counter on all packets with SYN=0.

  Reasoning: Such ACKs will be quite unusual, 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. And the impact of any error in the feedback on such ACKs will only be temporary.

3.2.2.3. 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 it is receiving continuous CE marking. Specifying exactly how to do this is beyond the scope of the present specification, but the sender might check whether the feedback for every packet it sends for the first three or four rounds indicates CE-marking. If continuous CE-marking is detected, for the remainder of the connection the Data Sender SHOULD NOT send ECN-capable packets and consequently it SHOULD NOT respond to any ECN feedback. The phrase ‘MUST NOT’ has been avoided to allow the sender to test whether it can resume sending ECN-capable packets. Throughout, it MUST remain in the AccECN feedback mode and it MUST continue to feed back any ECN markings on arriving packets (in its role as Data Receiver).

All the fall-back behaviours in this section 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.

3.2.2.4. 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.

3.2.2.5. 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

The following rules define when a Data Receiver in AccECN mode emits an ACK:

Change-Triggered ACKs: An AccECN Data Receiver SHOULD emit an ACK whenever a data packet marked CE arrives after the previous packet was not CE.
Even though this rule is stated as a "SHOULD", it is important for a transition to trigger an ACK if at all possible. The only valid exception to this rule is given below these bullets.

For the avoidance of doubt, this rule is deliberately worded to apply solely when _data_ packets arrive, but the comparison with the previous packet includes any packet, not just data packets.

Increment-Triggered ACKs: An AccECN Data Receiver MUST emit an ACK if 'n' CE marks have arrived since the previous ACK. If there is newly delivered data to acknowledge, 'n' SHOULD be 2. If there is no newly delivered data to acknowledge, 'n' SHOULD be 3 and MUST be no less than 3. In either case, 'n' MUST be no greater than 7.

The above 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.

If the arrivals of a number of data packets are all processed as one event, e.g. using large receive offload (LRO) or generic receive offload (GRO), both the above rules SHOULD be interpreted as requiring multiple ACKs to be emitted back-to-back (for each transition and for each repetition by 'n' CE marks). If this is problematic for high performance, either rule can be interpreted as requiring just a single ACK at the end of the whole receive event.

Even if a number of data packets do not arrive as one event, the 'Change-Triggered ACKs' rule could sometimes cause the ACK rate to be problematic for high performance (although high performance protocols such as DCTCP already successfully use change-triggered ACKs). The rationale for change-triggered ACKs is so that the Data Sender can rely on them to detect queue growth as soon as possible, particularly at the start of a flow. The 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 CE marks are infrequent, as is the case for most AQMs at the time of writing, or there are multiple marks in a row, the additional load will be low. However, marking patterns with numerous non-contiguous CE marks could increase the load significantly. 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.

With ECN-capable pure ACKs [I-D.ietf-tcpm-generalized-ecn], the 'Increment-Triggered ACKs' rule could cause ECN-marked pure ACKs to trigger further ACKs. Although TCP normally only ACKs newly delivered data, in this case the ACKs of ACKs would feed back new congestion state. The minimum of 3 for 'n' in this case ensures
that, even if there is pathological congestion in both directions, any resulting ping-pong of ACKs will be rapidly damped.

These ACKs of ACKs could be misidentified as duplicate ACKs in certain circumstances described below. Therefore, a host in AccECN mode that is sending ECN-capable pure ACKs SHOULD add one of the following additional checks when it tests whether an incoming pure ACK is a duplicate:

- If SACK has been negotiated for the connection, but there is no SACK option on the incoming pure ACK, it is not a duplicate;
- If timestamps are in use, and the incoming pure ACK echoes a timestamp older than the oldest unacknowledged data, it is not a duplicate.

In the unlikely event that neither SACK nor timestamps are in use, or if the implementation has opted not to include either of the above two checks, it SHOULD NOT send ECN-capable pure ACKs. If it does, it could lead to false detection of duplicate ACKs, causing spurious retransmission(s) with a resulting unnecessary reduction in congestion window; but only in certain circumstances. Specifically, if TCP peer A has been sending data, then receiving, then within one round trip it starts sending again, and the ECN-capable pure ACKs it sent in the previous round encounter heavy enough congestion to trigger peer B to invoke the above ‘n’-CE-mark rule. Also note that falsely considering these ACKs as duplicates would incorrectly imply that data left the network.

### 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  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.4.
Whenever a Data Receiver sends an AccECN Option, the rules in Section 3.2.3.3 allow it to omit unchanged fields from the tail of the option, to help cope with option space limitations, 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:

<table>
<thead>
<tr>
<th>Length</th>
<th>Type 0</th>
<th>Type 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>EE0B, ECEB, EE1B</td>
<td>EE1B, ECEB, EE0B</td>
</tr>
<tr>
<td>8</td>
<td>EE0B, ECEB</td>
<td>EE1B, ECEB</td>
</tr>
<tr>
<td>5</td>
<td>EE0B</td>
<td>EE1B</td>
</tr>
<tr>
<td>2</td>
<td>(empty)</td>
<td>(empty)</td>
</tr>
</tbody>
</table>

Fields included in AccECN TCP Options of each length and type

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, treating it as padding.

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.elb).

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 normally decodes 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.elb) 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.elb (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 byte counters held locally for each ECN marking 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. If there is somehow an AccECN Option on a SYN, it MUST be ignored when forwarded or received. (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 omit the AccECN Option in any of the above three cases due to insufficient option space or 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. [RFC9040] 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 3.2.2.4

Section 3.2.1 required the Data Receiver to initialize the r.e0b and r.e1b counters to a non-zero value. Therefore, in either direction the initial value of the EE0B field or EE1B field in the AccECN Option (if one exists) ought to be non-zero. If AccECN has been negotiated:

- the TCP server MAY check that the initial value of the EE0B field or the EE1B field is non-zero in the first segment that acknowledges sequence space that at least covers the ISN plus 1.
If it runs a test and either initial value 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 or the EE1B field is non-zero on the SYN/ACK. If it runs a test and either initial value 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 ought to provide more unambiguous feedback. However, it supplements but does not replace the ACE field. An endpoint using AccECN feedback MUST always reconcile the information provided in the ACE field with that in any AccECN Option, so that the state of the ACE-related packet counter can be relied on if future feedback does not carry the AccECN Option.

If the AccECN option is present, the s.cep counter might increase more than expected from the increase of the s.celb counter (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.celb counter has increased but the s.cep has not (and by testing ACK coverage it is certain how much the ACE field has wrapped), and if there is no explanation other than an invalid protocol transition due to some form of feedback mangling,
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 a Data Receiver in AccECN mode intends to use the AccECN TCP Option to provide feedback, the rules below determine when it includes an AccECN TCP Option, and which fields to include, given other options might be competing for limited option space:

Importance of Congestion Control: AccECN is for congestion control, which SHOULD generally be considered important relative to other TCP options.

If SACK has been negotiated, and the smallest recommended 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.

Recommended Simple Scheme: The Data Receiver SHOULD include an AccECN TCP Option on every scheduled ACK if any byte counter has incremented since the last ACK. Whenever possible, it SHOULD include a field for every byte counter that has changed at some time during the connection (see examples later).

A scheduled ACK means an ACK that the Data Receiver would send by its regular delayed ACK rules. Recall that Section 1.3 defines an 'ACK' as either with data payload or without. But the above rule is worded so that, in the common case when most of the data is from a server to a client, the server only includes an AccECN TCP Option while it is acknowledging data from the client.

When available TCP option space is limited on particular packets, the recommended scheme will need to include compromises. To guide the implementer the rules below are ranked in order of importance, but the final decision has to be implementation-dependent, because tradeoffs will alter as new TCP options are defined and new use-cases arise.

Necessary Option Length: The Data Receiver MUST only include an AccECN TCP Option on a packet if it includes all the counter(s) that have incremented since the previous AccECN Option. It MUST only truncate unchanged fields from the right-hand tail of the option to preserve the order of the remaining fields (see Section 3.2.3);
Change-Triggered AccECN TCP Options: If an arriving packet increments a different byte counter to that incremented by the previous packet, the Data Receiver SHOULD feed it back in an AccECN Option on the next scheduled ACK.

For the avoidance of doubt, this rule does not concern the arrival of control packets with no payload, because they cannot alter any byte counters.

Continual Repetition: Otherwise, if arriving packets continue to increment the same byte counter:

* the Data Receiver SHOULD include a counter that has continued to increment on the next scheduled ACK following a change-triggered AccECN TCP Option;

* 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.

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 recommended scheme is intended as a simple way to ensure that all the relevant byte counters will be carried on any ACK that reaches the Data Sender, no matter how many pure ACKs are filtered or coalesced along the network path, and without consuming the space available for payload data with counter field(s) that have never changed.

As an example of the recommended scheme, if ECT(0) is the only codepoint that has ever arrived in the IP-ECN field, the Data Receiver will feed back an AccECN0 TCP Option with only the EE0B field on every packet. However, as soon as even one CE-marked packet arrives, on every packet that acknowledges new data it will start to include an option with two fields, EE0B and ECEB. As a second example, if the first packet to arrive happens to be CE-marked, the Data Receiver will have to arbitrarily choose whether to precede the ECEB field with an EE0B field or an EE1B field. If it chooses, say,
EEB0 but it turns out never to receive ECT(0), it can start sending EE1B and ECEB instead - it does not have to include the EE0B field if the r.e0b counter has never changed during the connection.

With the recommended scheme, if the data sending direction switches during a connection, there can be cases where the AccECN TCP Option that is meant to feed back the counter values at the end of a volley in one direction never reaches the other peer, due to packet loss. ACE feedback ought to be sufficient to fill this gap, given accurate feedback becomes moot after data transmission has paused.

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 any of the rules in this section.

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 Transparent Middleboxes and 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 interpretation of the TCP specifications that is also not always up to date.

A middlebox that is not normalizing the TCP protocol and does not itself act as a back-to-back pair of TCP endpoints (i.e. a middlebox that intends to be transparent or invisible at the transport layer) ought to 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 match those in Section 3.2.1. This is because blocking apparently invalid values prevents the standardized set of values being extended in future (given outdated normalizers would block updated hosts from using the extended AccECN standard).
A TCP normalizer is likely to block or alter an AccECN TCP Option if the length value or the initial values of its byte-counter fields do not match one of those specified in Section 3.2.3 or Section 3.2.1. However, to comply with the present AccECN specification, a middlebox MUST NOT change the ACE field; or those fields of the AccECN Option that are currently specified in Section 3.2.3; or any AccECN field covered by integrity protection (e.g. [RFC5925]).

3.3.3. Requirements for TCP ACK Filtering

A node that implements ACK filtering (aka. thinning or coalescing) SHOULD determine if an ACK is part of a connection using AccECN and SHOULD then preserve the correct operation of AccECN feedback. The following notes might help with each part of this requirement:

- To determine whether a pure TCP ACK is part of an AccECN connection without resorting to connection tracking and per-flow state, a useful heuristic would be to check for a non-zero ECN field at the IP layer (because the ECN++ experiment only allows TCP pure ACKs to be ECN-capable if AccECN has been negotiated [I-D.ietf-tcpm-generalized-ecn]). This heuristic is simple and stateless. However, it might omit some AccECN ACKs, because it is only recommended but not obligatory to use ECN++ with AccECN - only deployment experience will tell. Also, TCP ACKs might be ECN-capable owing to some scheme other than AccECN, e.g. [RFC5690] or some future standards action. Again, only deployment experience will tell.

- The main concern with preserving correct AccECN operation involves leaving enough ACKs for the data sender to work out whether the 3-bit ACE field has wrapped. ACE field wrap is of less concern if packets also carry the AccECN TCP Option.

Note that the present specification of AccECN in TCP does not presume to rely on any of the above ACK filtering behaviour in the network (hence the use of 'SHOULD' rather than 'MUST' above), because it has to be robust against pre-existing network nodes that do not distinguish AccECN ACKs, and robust against ACK loss during overload more generally.

Section 5.2.1 of BCP 69 [RFC3449] gives best current practice on pure TCP ACK filtering. It gives no advice on ACKs carrying ECN feedback, other than that filtering ought to preserve the correct operation of ECN feedback, because at the time it said that "SACK and ECN remain areas of ongoing research". This section updates that best current practice 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 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, 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].
o 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 relative to other options, and which fields are more important to include.

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

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 an experimental change to the Data Sender that would be 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 standards track 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 protocol. 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:
TCP header flag reassignment

[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:

<table>
<thead>
<tr>
<th>Kind</th>
<th>Length</th>
<th>Meaning</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD0</td>
<td>N</td>
<td>Accurate ECN Order 0 (AccECN0)</td>
<td>RFC XXXX</td>
</tr>
<tr>
<td>TBD1</td>
<td>N</td>
<td>Accurate ECN Order 1 (AccECN1)</td>
<td>RFC XXXX</td>
</tr>
</tbody>
</table>

New TCP Option assignments

[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 present, 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 feedback 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. Assuring that Data Senders respond appropriately to ECN feedback is possible, but the scope of the present document is confined to the feedback protocol, and excludes the response to this feedback.

In Section 3.2.3 a Data Sender is allowed to ignore an unrecognized TCP AccECN Option length and read as many whole 3-octet fields from it as possible up to a maximum of 3, treating the remainder as padding. This opens up a potential covert channel of up to 29B (40 - (2+3*3))B. However, it is really an overt channel (not hidden) and it is no different to the use of unknown TCP options with unknown option lengths in general. Therefore, where this is of concern, it can already be adequately mitigated by regular TCP normalizer technology (see Section 3.3.2).

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. A covert channel can be used to compromise privacy. However, as explained above, undefined TCP options in general open up such channels and common techniques are available to close them off.
There is a potential concern that a Data Receiver could deliberately omit the AccECN Option pretending that it had been stripped by a middlebox. No known way can yet be contrived for a receiver to take advantage of this behaviour, which seems to always degrade its own performance. However, the concern is mentioned here for completeness.

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, Neal Cardwell, Yoshifumi Nishida, Martin Duke and Jonathan Morton 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. 15.0269. 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

11.2. Informative References


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

\[
ECEB = r.ceb \mod DIVOPT
\]

where \(\mod\) 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

\[
\begin{align*}
    s.ceb \mod DIVOPT &= 1 \\
    d.ceb &= (1461 + 2^{24} - 1) \mod 2^{24} \\
    &= 1460 \\
    s.ceb &= 33,554,433 + 1460 \\
    &= 33,555,893
\end{align*}
\]

In practice an implementation might use heuristics to guess the feedback in missing ACKs, then when it subsequently receives feedback it might find that it needs to correct its earlier heuristics as part of the decoding process. The above decoding process does not include any such heuristics.

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:

\[ \text{DIVACE} = 2^3 \]

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:

\[
\text{if } ((\text{newlyAckedB} > 0) \lor (\text{newlyAckedT} > 0)) \\
n.\text{cep} = (\text{ACE} + \text{DIVACE} - (s.\text{cep} \mod \text{DIVACE})) \mod \text{DIVACE}
\]

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 sequence 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

\[
n.\text{safer.cep} = \text{newlyAckedPkt} - ((\text{newlyAckedPkt} - n.\text{cep}) \mod \text{DIVACE})
\]

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 $10 - ((10-2) \% 8) = 10$).

Note that checks would need to be added to the above pseudocode for $(d.\text{cep} > \text{newlyAcknowledgedPkt})$, which could occur if $\text{newlyAcknowledgedPkt}$ had been wrongly estimated using an inappropriate packet size.

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 $\text{dSafer.cep}$ rather than $d.\text{cep}$ if the moving average of $\text{newlyAcknowledgedPkt}$ was well below 8.

Implementers could build in more heuristics to estimate prevailing average segment size and prevailing ECN marking. For instance, $\text{newlyAcknowledgedPkt}$ in the above formula could be replaced with $\text{newlyAcknowledgedPktHeur} = \text{newlyAcknowledgedPkt} \times p \times \text{MSS} / s$, 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 $\text{dSafer.cep}$ 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:

$$s.\text{cep} += \text{dSafer.cep}$$

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 \texttt{dSafer.cep} is different from \texttt{d.cep}, it can determine whether \texttt{d.cep} is likely to be a safe enough estimate by checking whether the average marked segment size ($s = \texttt{d.ceb}/\texttt{d.cep}$) is less than the MSS (where \texttt{d.ceb} is the amount of newly CE-marked bytes - see Appendix A.1). Specifically, it could use the following algorithm:

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

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

\begin{center}
\begin{tabular}{c|c|c}
\texttt{sSafer} & \texttt{MSS} & \texttt{dSafer.cep} is safest \texttt{MSS/SAFETY_FACTOR}\texttt{d.cep is safe \texttt{d.cep}}
\hline
\end{tabular}
\end{center}

\begin{center}
\begin{tabular}{c|c|c|c|c|c}
& & & & & \\
\hline
\texttt{MSS/SAFETY_FACTOR} & \texttt{-------------------} & \texttt{s}
\hline
\end{tabular}
\end{center}

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

- if \texttt{d.cep}=0, \texttt{dSafer.cep}=8 and \texttt{d.ceb}=1460, then \texttt{s}=infinity and \texttt{sSafer}=182.5.
  Therefore even though the average size of 8 data segments is unlikely to have been as small as \texttt{MSS}/8, \texttt{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:

\[ s_\text{ave} = \frac{\text{flightsize}}{\text{packets_in_flight}}, \]

where flightsize is the variable that TCP already maintains for the number of bytes in flight. To avoid floating point arithmetic, it could right-bit-shift by \( \log_2(\text{packets_in_flight}) \), where \( \log() \) means log base 2.

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

\[ s_\text{ave} = a * s + (1-a) * s_\text{ave}, \]

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 Count Not-ECT Bytes

A Data Sender in AccECN mode can infer the amount of TCP payload data arriving at the receiver marked Not-ECT from the difference between the amount of newly ACKed data and the sum of the bytes with the other three markings, d.ceb, d.e0b and d.elb.

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 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 3.2.2.4 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
HyStart++: Modified Slow Start for TCP
draft-ietf-tcpm-hystartplusplus-04

Abstract

This document describes HyStart++, a simple modification to the slow start phase of TCP congestion control algorithms. Traditional slow start can overshoot the ideal send rate in many cases, causing high packet loss and poor performance. HyStart++ uses a delay increase heuristic to find an exit point before possible overshoot. It also adds a mitigation to prevent jitter from causing premature slow start exit.

Status of This Memo

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

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

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

This Internet-Draft will expire on 27 July 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
1. Introduction

[RFC5681] describes the slow start congestion control algorithm for TCP. The slow start algorithm is used when the congestion window (cwnd) is less than the slow start threshold (ssthresh). During slow start, in absence of packet loss signals, TCP increases cwnd exponentially to probe the network capacity. This fast growth can overshoot the ideal sending rate and cause significant packet loss which cannot always be recovered efficiently.

HyStart++ uses delay increase as a signal to exit slow start before potential packet loss occurs as a result of overshoot. This is one of two algorithms specified in [HyStart]. After the slow start exit, a novel Conservative Slow Start (CSS) phase is used to determine whether the slow start exit was premature and to resume slow start. This mitigation improves performance in presence of jitter.

HyStart++ reduces packet loss and retransmissions, and improves goodput in lab measurements and real world deployments.
2. Terminology

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

3. Definitions

We repeat here some definition from [RFC5681] to aid the reader.

SENDER MAXIMUM SEGMENT SIZE (SMSS): The SMSS is the size of the largest segment that the sender can transmit. This value can be based on the maximum transmission unit of the network, the path MTU discovery [RFC1191, RFC4821] algorithm, RMSS (see next item), or other factors. The size does not include the TCP/IP headers and options.

RECEIVER MAXIMUM SEGMENT SIZE (RMSS): The RMSS is the size of the largest segment the receiver is willing to accept. This is the value specified in the MSS option sent by the receiver during connection startup. Or, if the MSS option is not used, it is 536 bytes [RFC1122]. The size does not include the TCP/IP headers and options.

RECEIVER WINDOW (rwnd): The most recently advertised receiver window.

CONGESTION WINDOW (cwnd): A TCP state variable that limits the amount of data a TCP can send. At any given time, a TCP MUST NOT send data with a sequence number higher than the sum of the highest acknowledged sequence number and the minimum of cwnd and rwnd.

4. HyStart++ Algorithm

4.1. Summary

[HyStart] specifies two algorithms (a "Delay Increase" algorithm and an "Inter-Packet Arrival" algorithm) to be run in parallel to detect that the sending rate has reached capacity. In practice, the Inter-Packet Arrival algorithm does not perform well and is not able to detect congestion early, primarily due to ACK compression. The idea of the Delay Increase algorithm is to look for spikes in RTT (round-trip time), which suggest that the bottleneck buffer is filling up.

In HyStart++, a TCP sender uses traditional slow start and then uses the "Delay Increase" algorithm to trigger an exit from slow start. But instead of going straight from slow start to congestion avoidance, the sender spends a number of RTTs in a Conservative Slow Start (CSS) phase to determine whether the exit from slow start was premature. During CSS, the congestion window is grown exponentially...
like in regular slow start, but with a smaller exponential base, resulting in less aggressive growth. If the RTT reduces during CSS, it’s concluded that the RTT spike was not related to congestion caused by the connection sending at a rate greater than the ideal send rate, and the connection resumes slow start. If the RTT inflation persists throughout CSS, the connection enters congestion avoidance.

4.2. Algorithm Details

For the pseudocode, we assume that Appropriate Byte Counting (as described in [RFC3465]) is in use and \( L \) is the cwnd increase limit as discussed in RFC 3465.

lastRoundMinRTT and currentRoundMinRTT are initialized to infinity at the initialization time.

Hystart++ measures rounds using sequence numbers, as follows:

1. Define windowEnd as a sequence number initialized to SND.UNA
2. When windowEnd is ACKed, the current round ends and windowEnd is set to SND.NXT

At the start of each round during standard slow start ([RFC5681]) and CSS:

1. \( \text{lastRoundMinRTT} = \text{currentRoundMinRTT} \)
2. \( \text{currentRoundMinRTT} = \infty \)
3. \( \text{rttSampleCount} = 0 \)

For each arriving ACK in slow start, where \( N \) is the number of previously unacknowledged bytes acknowledged in the arriving ACK:

1. Update the cwnd
2. \( \text{cwnd} = \text{cwnd} + \min(N, L \times \text{SMSS}) \)
3. Keep track of minimum observed RTT
4. \( \text{currentRoundMinRTT} = \min(\text{currentRoundMinRTT}, \text{currRTT}) \)
5. where \( \text{currRTT} \) is the RTT sampled from the latest incoming ACK
6. \( \text{rttSampleCount} += 1 \)
For rounds where \( N_{\text{RTT SAMPLE}} \) RTT samples have been obtained and currentRoundMinRTT and lastRoundMinRTT are valid, check if delay increase triggers slow start exit

- if \((\text{rttSampleCount} \geq N_{\text{RTT SAMPLE}} \text{ AND currentRoundMinRTT } \neq \infty \text{ AND lastRoundMinRTT } \neq \infty)\)
  - \( \text{RttThresh} = \text{clamp}(MIN_{\text{RTT THRESH}}, \text{lastRoundMinRTT} / 8, MAX_{\text{RTT THRESH}}) \)
  - if \((\text{currentRoundMinRTT} \geq (\text{lastRoundMinRTT} + \text{RttThresh}))\)
    + \( \text{cssBaselineMinRtt} = \text{currentRoundMinRTT} \)
    + exit slow start and enter CSS

CSS lasts at most CSS_ROUNDS rounds. If the transition into CSS happens in the middle of a round, that partial round counts towards the limit.

For each arriving ACK in CSS, where \( N \) is the number of previously unacknowledged bytes acknowledged in the arriving ACK:

Update the cwnd

- \( \text{cwnd} = \text{cwnd} + (\min (N, L * \text{SMSS}) / \text{CSS}_\text{GROWTH}_\text{DIVISOR}) \)

Keep track of minimum observed RTT

- \( \text{currentRoundMinRTT} = \min(\text{currentRoundMinRTT}, \text{currRTT}) \)
  - where \( \text{currRTT} \) is the sampled RTT from the incoming ACK
  - \( \text{rttSampleCount} += 1 \)

For CSS rounds where \( N_{\text{RTT SAMPLE}} \) RTT samples have been obtained, check if current round's minRTT drops below baseline indicating that HyStart exit was spurious.

- if \((\text{currentRoundMinRTT} < \text{cssBaselineMinRtt})\)
  - \( \text{cssBaselineMinRtt} = \infty \)
  - resume slow start including HyStart++

If CSS_ROUNDS rounds are complete, enter congestion avoidance.

* \( \text{ssthresh} = \text{cwnd} \)
If loss or ECN-marking is observed anytime during standard slow start or CSS, enter congestion avoidance.

* ssthresh = cwnd

4.3. Tuning constants

It is RECOMMENDED that a HyStart++ implementation use the following constants:

* MIN_RTT_THRESH = 4 msec
* MAX_RTT_THRESH = 16 msec
* N_RTT_SAMPLE = 8
* CSS_GROWTH_DIVISOR = 4
* CSS_ROUNDS = 5

These constants have been determined with lab measurements and real world deployments. An implementation MAY tune them for different network characteristics.

The delay increase sensitivity is determined by MIN_RTT_THRESH and MAX_RTT_THRESH. Smaller values of MIN_RTT_THRESH may cause spurious exits from slow start. Larger values of MAX_RTT_THRESH may result in slow start not exiting until loss is encountered for connections on large RTT paths.

A TCP implementation is required to take at least one RTT sample each round. Using lower values of N_RTT_SAMPLE will lower the accuracy of the measured RTT for the round; higher values will improve accuracy at the cost of more processing.

The minimum value of CSS_GROWTH_DIVISOR MUST be at least 2. A value of 1 results in the same aggressive behavior as regular slow start. Values larger than 4 will cause the algorithm to be less aggressive and maybe less performant.

Smaller values of CSS_ROUNDS may miss detecting jitter and larger values may limit performance.

An implementation SHOULD use HyStart++ only for the initial slow start (when ssthresh is at its initial value of arbitrarily high per [RFC5681]) and fall back to using traditional slow start for the remainder of the connection lifetime. This is acceptable because subsequent slow starts will use the discovered ssthresh value to exit
slow start and avoid the overshoot problem. An implementation MAY use HyStart++ to grow the restart window ([RFC5681]) after a long idle period.

5. Deployments and Performance Evaluations

As of the time of writing, HyStart++ as described in draft versions 01 through 04 was default enabled for all TCP connections in the Windows operating system for over three years. The original Hystart has been default-enabled for all TCP connections in the Linux operating system using the default congestion control module CUBIC ([RFC8312]) for a decade.

In lab measurements with Windows TCP, HyStart++ shows both goodput improvements as well as reductions in packet loss and retransmissions. For example across a variety of tests on a 100 Mbps link with a bottleneck buffer size of bandwidth-delay product, HyStart++ reduces bytes retransmitted by 50% and retransmission timeouts by 36%.

In an A/B test for HyStart++ draft 01 across a large Windows device population, out of 52 billion TCP connections, 0.7% of connections move from 1 RTO to 0 RTOs and another 0.7% connections move from 2 RTOs to 1 RTO with HyStart++. This test did not focus on send heavy connections and the impact on send heavy connections is likely much higher. We plan to conduct more such production experiments to gather more data in the future.

6. Security Considerations

HyStart++ enhances slow start and inherits the general security considerations discussed in [RFC5681].

7. IANA Considerations

This document has no actions for IANA.

8. References

8.1. Normative References


Balasubramanian, et al. Expires 27 July 2022
8.2. Informative References


Authors’ Addresses

Praveen Balasubramanian
Microsoft
One Microsoft Way
Redmond, WA 98052
United States of America
Phone: +1 425 538 2782
Email: pravb@microsoft.com

Yi Huang
Microsoft
Phone: +1 425 703 0447
Email: huanyi@microsoft.com

Matt Olson
Microsoft
Phone: +1 425 538 8598
Email: maolson@microsoft.com
CUBIC for Fast and Long-Distance Networks
draft-ietf-tcpm-rfc8312bis-07

Abstract

CUBIC is a standard TCP congestion control algorithm that uses a cubic function instead of a linear congestion window increase function to improve scalability and stability over fast and long-distance networks. CUBIC has been adopted as the default TCP congestion control algorithm by the Linux, Windows, and Apple stacks.

This document updates the specification of CUBIC to include algorithmic improvements based on these implementations and recent academic work. Based on the extensive deployment experience with CUBIC, it also moves the specification to the Standards Track, obsoleting RFC 8312. This also requires updating RFC 5681, to allow for CUBIC’s occasionally more aggressive sending behavior.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-tcpm-rfc8312bis/.

Discussion of this document takes place on the TCPM Working Group mailing list (mailto:tcpm@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tcpm/.

Source for this draft and an issue tracker can be found at https://github.com/NTAP/rfc8312bis.

Note to the RFC Editor
xml2rfc currently renders <em></em> in the XML by surrounding the corresponding text with underscores. This is highly distracting; please manually remove the underscores when doing the final edits to the text version of this document.

(There is an issue open against xml2rfc to stop doing this in the future: https://trac.tools.ietf.org/tools/xml2rfc/trac/ticket/596)

Also, please manually change "Figure" to "Equation" for all artwork with anchors beginning with "eq" - xml2rfc doesn’t seem to be able to do this.

Status of This Memo

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

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

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

This Internet-Draft will expire on 5 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction ............................................. 4
2. Conventions ............................................ 5
3. Design Principles of CUBIC ............................. 5

Xu, et al. Expires 5 September 2022
1. Introduction

CUBIC has been adopted as the default TCP congestion control algorithm in the Linux, Windows, and Apple stacks, and has been used and deployed globally. Extensive, decade-long deployment experience in vastly different Internet scenarios has convincingly demonstrated that CUBIC is safe for deployment on the global Internet and delivers substantial benefits over classical Reno congestion control [RFC5681]. It is therefore to be regarded as the currently most widely deployed standard for TCP congestion control. CUBIC can also be used for other transport protocols such as QUIC [RFC9000] and SCTP [RFC4960] as a default congestion controller.

The design of CUBIC was motivated by the well-documented problem classical Reno TCP has with low utilization over fast and long-distance networks [K03][RFC3649]. This problem arises from a slow increase of the congestion window following a congestion event in a network with a large bandwidth-delay product (BDP). [HLRX07] indicates that this problem is frequently observed even in the range of congestion window sizes over several hundreds of packets. This problem is equally applicable to all Reno-style standards and their variants, including TCP-Reno [RFC5681], TCP-NewReno [RFC6582][RFC6675], SCTP [RFC4960], TFRC [RFC5348], and QUIC congestion control [RFC9002], which use the same linear increase function for window growth. We refer to all Reno-style standards and their variants collectively as "Reno" below.

CUBIC, originally proposed in [HRX08], is a modification to the congestion control algorithm of classical Reno to remedy this problem. Specifically, CUBIC uses a cubic function instead of the linear window increase function of Reno to improve scalability and stability under fast and long-distance networks.

This document updates the specification of CUBIC to include algorithmic improvements based on the Linux, Windows, and Apple implementations and recent academic work. Based on the extensive deployment experience with CUBIC, it also moves the specification to the Standards Track, obsoleting [RFC8312]. This requires an update to [RFC5681], which limits the aggressiveness of Reno TCP implementations in its Section 3. Since CUBIC is occasionally more aggressive than the [RFC5681] algorithms, this document updates [RFC5681] to allow for CUBIC’s behavior.
Binary Increase Congestion Control (BIC-TCP) [XHR04], a predecessor of CUBIC, was selected as the default TCP congestion control algorithm by Linux in the year 2005 and had been used for several years by the Internet community at large.

CUBIC uses a similar window increase function as BIC-TCP and is designed to be less aggressive and fairer to Reno in bandwidth usage than BIC-TCP while maintaining the strengths of BIC-TCP such as stability, window scalability, and round-trip time (RTT) fairness.

In the following sections, we first briefly explain the design principles of CUBIC, then provide the exact specification of CUBIC, and finally discuss the safety features of CUBIC following the guidelines specified in [RFC5033].

2. Conventions

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.

3. Design Principles of CUBIC

CUBIC is designed according to the following design principles:

Principle 1: For better network utilization and stability, CUBIC uses both the concave and convex profiles of a cubic function to increase the congestion window size, instead of using just a convex function.

Principle 2: To be Reno-friendly, CUBIC is designed to behave like Reno in networks with short RTTs and small bandwidth where Reno performs well.

Principle 3: For RTT-fairness, CUBIC is designed to achieve linear bandwidth sharing among flows with different RTTs.

Principle 4: CUBIC appropriately sets its multiplicative window decrease factor in order to balance between the scalability and convergence speed.
3.1. Principle 1 for the CUBIC Increase Function

For better network utilization and stability, CUBIC [HRX08] uses a cubic window increase function in terms of the elapsed time from the last congestion event. While most alternative congestion control algorithms to Reno increase the congestion window using convex functions, CUBIC uses both the concave and convex profiles of a cubic function for window growth.

After a window reduction in response to a congestion event detected by duplicate ACKs, Explicit Congestion Notification-Echo (ECN-Echo, ECE) ACKs [RFC3168], TCP RACK [RFC8985] or QUIC loss detection [RFC9002], CUBIC remembers the congestion window size at which it received the congestion event and performs a multiplicative decrease of the congestion window. When CUBIC enters into congestion avoidance, it starts to increase the congestion window using the concave profile of the cubic function. The cubic function is set to have its plateau at the remembered congestion window size, so that the concave window increase continues until then. After that, the cubic function turns into a convex profile and the convex window increase begins.

This style of window adjustment (concave and then convex) improves the algorithm stability while maintaining high network utilization [CEHRX09]. This is because the window size remains almost constant, forming a plateau around the remembered congestion window size of the last congestion event, where network utilization is deemed highest. Under steady state, most window size samples of CUBIC are close to that remembered congestion window size, thus promoting high network utilization and stability.

Note that congestion control algorithms that only use convex functions to increase the congestion window size have their maximum increments around the remembered congestion window size of the last congestion event, and thus introduce many packet bursts around the saturation point of the network, likely causing frequent global loss synchronizations.

3.2. Principle 2 for Reno-Friendliness

CUBIC promotes per-flow fairness to Reno. Note that Reno performs well over paths with short RTTs and small bandwidths (or small BDPs). There is only a scalability problem in networks with long RTTs and large bandwidths (or large BDPs).

A congestion control algorithm designed to be friendly to Reno on a per-flow basis must increase its congestion window less aggressively in small BDP networks than in large BDP networks.
The aggressiveness of CUBIC mainly depends on the maximum window size before a window reduction, which is smaller in small-BDP networks than in large-BDP networks. Thus, CUBIC increases its congestion window less aggressively in small-BDP networks than in large-BDP networks.

Furthermore, in cases when the cubic function of CUBIC would increase the congestion window less aggressively than Reno, CUBIC simply follows the window size of Reno to ensure that CUBIC achieves at least the same throughput as Reno in small-BDP networks. We call this region where CUBIC behaves like Reno the "Reno-friendly region".

3.3. Principle 3 for RTT Fairness

Two CUBIC flows with different RTTs have a throughput ratio that is linearly proportional to the inverse of their RTT ratio, where the throughput of a flow is approximately the size of its congestion window divided by its RTT.

Specifically, CUBIC maintains a window increase rate independent of RTTs outside the Reno-friendly region, and thus flows with different RTTs have similar congestion window sizes under steady state when they operate outside the Reno-friendly region.

This notion of a linear throughput ratio is similar to that of Reno under high statistical multiplexing where packet loss is independent of individual flow rates. However, under low statistical multiplexing, the throughput ratio of Reno flows with different RTTs is quadratically proportional to the inverse of their RTT ratio [XHR04].

CUBIC always ensures a linear throughput ratio independent of the amount of statistical multiplexing. This is an improvement over Reno. While there is no consensus on particular throughput ratios for different RTT flows, we believe that over wired Internet paths, use of a linear throughput ratio seems more reasonable than equal throughputs (i.e., the same throughput for flows with different RTTs) or a higher-order throughput ratio (e.g., a quadratical throughput ratio of Reno under low statistical multiplexing environments).

3.4. Principle 4 for the CUBIC Decrease Factor

To balance between scalability and convergence speed, CUBIC sets the multiplicative window decrease factor to 0.7, whereas Reno uses 0.5.

While this improves the scalability of CUBIC, a side effect of this decision is slower convergence, especially under low statistical multiplexing. This design choice is following the observation that
HighSpeed TCP (HSTCP) [RFC3649] and other approaches (e.g., [GV02]) made: the current Internet becomes more asynchronous with less frequent loss synchronizations under high statistical multiplexing.

In such environments, even strict Multiplicative-Increase Multiplicative-Decrease (MIMD) can converge. CUBIC flows with the same RTT always converge to the same throughput independent of statistical multiplexing, thus achieving intra-algorithm fairness. We also find that in environments with sufficient statistical multiplexing, the convergence speed of CUBIC is reasonable.

4. CUBIC Congestion Control

In this section, we discuss how the congestion window is updated during the different stages of the CUBIC congestion controller.

4.1. Definitions

The unit of all window sizes in this document is segments of the maximum segment size (MSS), and the unit of all times is seconds. Implementations can use bytes to express window sizes, which would require factoring in the maximum segment size wherever necessary and replacing _segments_acked_ with the number of bytes acknowledged in Figure 4.

4.1.1. Constants of Interest

__cubic_: CUBIC multiplicative decrease factor as described in Section 4.6.

__cubic_: CUBIC additive increase factor used in Reno-friendly region as described in Section 4.3.

_C_: constant that determines the aggressiveness of CUBIC in competing with other congestion control algorithms in high BDP networks. Please see Section 5 for more explanation on how it is set. The unit for _C_ is

\[
\text{segment} \quad \frac{1}{3} \quad \text{second}
\]

4.1.2. Variables of Interest

This section defines the variables required to implement CUBIC:
_RTT_: Smoothed round-trip time in seconds, calculated as described in [RFC6298].

_cwnd_: Current congestion window in segments.

_ssthresh_: Current slow start threshold in segments.

_W_max_: Size of _cwnd_ in segments just before _cwnd_ was reduced in the last congestion event when fast convergence is disabled. However, if fast convergence is enabled, the size may be further reduced based on the current saturation point.

_K_: The time period in seconds it takes to increase the congestion window size at the beginning of the current congestion avoidance stage to _W_max_.

_current_time_: Current time of the system in seconds.

_epoch_start_: The time in seconds at which the current congestion avoidance stage started.

_cwnd_start_: The _cwnd_ at the beginning of the current congestion avoidance stage, i.e., at time _epoch_start_.

W_cubic(_t_): The congestion window in segments at time _t_ in seconds based on the cubic increase function, as described in Section 4.2.

_target_: Target value of congestion window in segments after the next RTT, that is, W_cubic(_t_ + _RTT_), as described in Section 4.2.

_W_est_: An estimate for the congestion window in segments in the Reno-friendly region, that is, an estimate for the congestion window of Reno.

_segments_acked_: Number of MSS-sized segments acked when a "new ACK" is received, i.e., an ACK that cumulatively acknowledges the delivery of new data. This number will be a decimal value when a new ACK acknowledges an amount of data that is not MSS-sized. Specifically, it can be less than 1 when a new ACK acknowledges a segment smaller than the MSS.

4.2. Window Increase Function

CUBIC maintains the acknowledgment (ACK) clocking of Reno by increasing the congestion window only at the reception of a new ACK. It does not make any changes to the TCP Fast Recovery and Fast Retransmit algorithms [RFC6582][RFC6675].
During congestion avoidance, after a congestion event is detected by mechanisms described in Section 3.1, CUBIC uses a window increase function different from Reno.

CUBIC uses the following window increase function:

\[
W(t) = \frac{3}{C} (t - K) + W_{\text{max}}
\]

Figure 1

where \( t \) is the elapsed time in seconds from the beginning of the current congestion avoidance stage, that is,

\[ t = \text{current\_time} - \text{epoch\_start} \]

and where \( \text{epoch\_start} \) is the time at which the current congestion avoidance stage starts. \( K \) is the time period that the above function takes to increase the congestion window size at the beginning of the current congestion avoidance stage to \( W_{\text{max}} \) if there are no further congestion events and is calculated using the following equation:

\[
K = \left( \frac{W - \text{cwnd}}{\text{cwnd}_{\text{start}}} \right)^{\frac{3}{C}}
\]

Figure 2

where \( \text{cwnd}_{\text{start}} \) is the congestion window at the beginning of the current congestion avoidance stage.

Upon receiving a new ACK during congestion avoidance, CUBIC computes the \( \text{target\_congestion\_window\_size} \) after the next \( \text{RTT} \) using Figure 1 as follows, where \( \text{RTT} \) is the smoothed round-trip time. The lower and upper bounds below ensure that CUBIC’s congestion window increase rate is non-decreasing and is less than the increase rate of slow start [SXEZ19].
The elapsed time \( t \) in Figure 1 MUST NOT include periods during which \( _{\text{cwnd}} \) has not been updated due to application-limited behavior (see Section 5.8).

Depending on the value of the current congestion window size \( _{\text{cwnd}} \), CUBIC runs in three different regions:

1. The Reno-friendly region, which ensures that CUBIC achieves at least the same throughput as Reno.

2. The concave region, if CUBIC is not in the Reno-friendly region and \( _{\text{cwnd}} \) is less than \( _{\text{W_max}} \).

3. The convex region, if CUBIC is not in the Reno-friendly region and \( _{\text{cwnd}} \) is greater than \( _{\text{W_max}} \).

Below, we describe the exact actions taken by CUBIC in each region.

### 4.3. Reno-Friendly Region

Reno performs well in certain types of networks, for example, under short RTTs and small bandwidths (or small BDPs). In these networks, CUBIC remains in the Reno-friendly region to achieve at least the same throughput as Reno.

The Reno-friendly region is designed according to the analysis in [FHP00], which studies the performance of an AIMD algorithm with an additive factor of \( (\text{segments per } _{\text{RTT}}) \) and a multiplicative factor of \( _p \), denoted by \( \text{AIMD}(, , _p) \). \( _p \) is the packet loss rate.

Specifically, the average congestion window size of \( \text{AIMD}(, ) \) can be calculated using Figure 3.
By the same analysis, to achieve the same average window size as Reno that uses AIMD(1, 0.5), AVG_AIMD(, ) must be equal to,

\[
\text{AVG_AIMD}(x, y) = \left| \frac{1 - \frac{3 \cdot \frac{1}{1 + c}}{c + 1}}{2 \cdot (1 - x) \cdot y} \right|
\]

Figure 3

Thus, CUBIC uses Figure 4 to estimate the window size _W_est_ in the Reno-friendly region with

\[
\text{cubic} \cdot \frac{1 - \frac{3 \cdot \frac{1}{c + 1}}{c + 1}}{c + 1} = 3 \cdot \frac{1 - \frac{3 \cdot \frac{1}{c + 1}}{c + 1}}{c + 1}
\]

which achieves the same average window size as Reno. When receiving a new ACK in congestion avoidance (where _cwnd_ could be greater than or less than _W_max_), CUBIC checks whether _W_cubic(_t_) is less than _W_est_. If so, CUBIC is in the Reno-friendly region and _cwnd_ SHOULD be set to _W_est_ at each reception of a new ACK.

_W_est_ is set equal to _cwnd_start_ at the start of the congestion avoidance stage. After that, on every new ACK, _W_est_ is updated using Figure 4. Note that this equation is for a connection where Appropriate Byte Counting (ABC) [RFC3465] is disabled. For a connection with ABC enabled, this equation SHOULD be adjusted by using the number of acknowledged bytes instead of acknowledged segments. Also note that this equation works for connections with enabled or disabled Delayed ACKs [RFC5681], as _segments_acked_ will be different based on the segments actually acknowledged by a new ACK.

\[
_w = w + \frac{\text{segments}_\text{acked}}{\text{est} \cdot \text{est} \cdot \text{cubic} \cdot \text{cwnd}}
\]

Figure 4
Note that once $W_{est}$ reaches $W_{max}$, that is, $W_{est} \geq W_{max}$, CUBIC needs to start probing to determine the new value of $W_{max}$. At this point, $cubic$ SHOULD be set to 1 to ensure that CUBIC can achieve the same congestion window increment as Reno, which uses AIMD(1, 0.5).

4.4. Concave Region

When receiving a new ACK in congestion avoidance, if CUBIC is not in the Reno-friendly region and $cwnd$ is less than $W_{max}$, then CUBIC is in the concave region. In this region, $cwnd$ MUST be incremented by

$$\frac{target - cwnd}{cwnd}$$

for each received new ACK, where $target$ is calculated as described in Section 4.2.

4.5. Convex Region

When receiving a new ACK in congestion avoidance, if CUBIC is not in the Reno-friendly region and $cwnd$ is larger than or equal to $W_{max}$, then CUBIC is in the convex region.

The convex region indicates that the network conditions might have changed since the last congestion event, possibly implying more available bandwidth after some flow departures. Since the Internet is highly asynchronous, some amount of perturbation is always possible without causing a major change in available bandwidth.

Unless it is overridden by the AIMD window increase, CUBIC is very careful in this region. The convex profile aims to increase the window very slowly at the beginning when $cwnd$ is around $W_{max}$ and then gradually increases its rate of increase. We also call this region the "maximum probing phase", since CUBIC is searching for a new $W_{max}$. In this region, $cwnd$ MUST be incremented by

$$\frac{target - cwnd}{cwnd}$$

for each received new ACK, where $target$ is calculated as described in Section 4.2.
4.6. Multiplicative Decrease

When a congestion event is detected by mechanisms described in Section 3.1, CUBIC updates _W_max_ and reduces _cwnd_ and _ssthresh_ immediately as described below. In case of packet loss, the sender MUST reduce _cwnd_ and _ssthresh_ immediately upon entering loss recovery, similar to [RFC5681] (and [RFC6675]). Note that other mechanisms, such as Proportional Rate Reduction [RFC6937], can be used to reduce the sending rate during loss recovery more gradually. The parameter __cubic_ SHOULD be set to 0.7, which is different from the multiplicative decrease factor used in [RFC5681] (and [RFC6675]) during fast recovery.

In Figure 5, _flight_size_ is the amount of outstanding data in the network, as defined in [RFC5681]. Note that a rate-limited application with idle periods or periods when unable to send at the full rate permitted by _cwnd_ may easily encounter notable variations in the volume of data sent from one RTT to another, resulting in _flight_size_ that is significantly less than _cwnd_ on a congestion event. This may decrease _cwnd_ to a much lower value than necessary. To avoid suboptimal performance with such applications, the mechanisms described in [RFC7661] can be used to mitigate this issue as it would allow using a value between _cwnd_ and _flight_size_ to calculate the new _ssthresh_ in Figure 5. The congestion window growth mechanism defined in [RFC7661] is safe to use even when _cwnd_ is greater than the receive window as it validates _cwnd_ based on the amount of data acknowledged by the network in an RTT which implicitly accounts for the allowed receive window. Some implementations of CUBIC currently use _cwnd_ instead of _flight_size_ when calculating a new _ssthresh_ using Figure 5.

\[
\text{flight_size} \times \frac{\text{cubic}}{\max(\text{ssthresh}, 2)} \quad \text{// reduction on packet loss, cwnd is at least 2 MSS}
\]

\[
\text{cwnd} = \begin{cases} 
\max(\text{ssthresh}, 1) & \text{// reduction on ECE, cwnd is at least 1 MSS} \\
\max(\text{ssthresh}, 2) & \text{// ssthresh is at least 2 MSS}
\end{cases}
\]

\[
\text{ssthresh} = \frac{\text{flight_size}}{\max(\text{ssthresh}, 2)} \quad \text{// new ssthresh}
\]

Figure 5
A side effect of setting \texttt{cubic} to a value bigger than 0.5 is slower convergence. We believe that while a more adaptive setting of \texttt{cubic} could result in faster convergence, it will make the analysis of CUBIC much harder.

Note that CUBIC MUST continue to reduce \texttt{cwnd} in response to congestion events due to ECN-Echo ACKs until it reaches a value of 1 MSS. If congestion events indicated by ECN-Echo ACKs persist, a sender with a \texttt{cwnd} of 1 MSS MUST reduce its sending rate even further. It can achieve that by using a retransmission timer with exponential backoff, as described in [RFC3168].

4.7. Fast Convergence

To improve convergence speed, CUBIC uses a heuristic. When a new flow joins the network, existing flows need to give up some of their bandwidth to allow the new flow some room for growth, if the existing flows have been using all the network bandwidth. To speed up this bandwidth release by existing flows, the following "Fast Convergence" mechanism SHOULD be implemented.

With Fast Convergence, when a congestion event occurs, we update \texttt{W_max} as follows, before the window reduction as described in Section 4.6.

\[
W = \begin{cases} 
1 + \frac{\text{cubic if } \texttt{cwnd} < W \text{ and fast convergence is enabled}}{2} \times \text{max} \\
\text{cwnd} & \text{max} \\
\text{further reduce } W & \text{max} \\
\text{otherwise, remember } \texttt{cwnd} \text{ before reduction} & \text{cwnd}
\end{cases}
\]

At a congestion event, if the current \texttt{cwnd} is less than \texttt{W_max}, this indicates that the saturation point experienced by this flow is getting reduced because of a change in available bandwidth. Then we allow this flow to release more bandwidth by reducing \texttt{W_max} further. This action effectively lengthens the time for this flow to increase its congestion window, because the reduced \texttt{W_max} forces the flow to plateau earlier. This allows more time for the new flow to catch up to its congestion window size.
Fast Convergence is designed for network environments with multiple CUBIC flows. In network environments with only a single CUBIC flow and without any other traffic, Fast Convergence SHOULD be disabled.

4.8. Timeout

In case of a timeout, CUBIC follows Reno to reduce _cwnd_ [RFC5681], but sets _ssthresh_ using __cubic_ (same as in Section 4.6) in a way that is different from Reno TCP [RFC5681].

During the first congestion avoidance stage after a timeout, CUBIC increases its congestion window size using Figure 1, where _t_ is the elapsed time since the beginning of the current congestion avoidance, _K_ is set to 0, and _W_max_ is set to the congestion window size at the beginning of the current congestion avoidance stage. In addition, for the Reno-friendly region, _W_est_ SHOULD be set to the congestion window size at the beginning of the current congestion avoidance.

4.9. Spurious Congestion Events

In cases where CUBIC reduces its congestion window in response to having detected packet loss via duplicate ACKs or timeouts, there is a possibility that the missing ACK would arrive after the congestion window reduction and a corresponding packet retransmission. For example, packet reordering could trigger this behavior. A high degree of packet reordering could cause multiple congestion window reduction events, where spurious losses are incorrectly interpreted as congestion signals, thus degrading CUBIC’s performance significantly.

For TCP, there are two types of spurious events - spurious timeouts and spurious fast retransmits. In case of QUIC, there are no spurious timeouts as the loss is only detected after receiving an ACK.

4.9.1. Spurious timeout

An implementation MAY detect spurious timeouts based on the mechanisms described in Forward RTO-Recovery [RFC5682]. Experimental alternatives include Eifel [RFC3522]. When a spurious timeout is detected, a TCP implementation MAY follow the response algorithm described in [RFC4015] to restore the congestion control state and adapt the retransmission timer to avoid further spurious timeouts.
4.9.2. Spurious loss detected by acknowledgements

Upon receiving an ACK, a TCP implementation MAY detect spurious losses either using TCP Timestamps or via D-SACK[RFC2883]. Experimental alternatives include Eifel detection algorithm [RFC3522] which uses TCP Timestamps and D-SACK based detection [RFC3708] which uses D-SACK information. A QUIC implementation can easily determine a spurious loss if a QUIC packet is acknowledged after it has been marked as lost and the original data has been retransmitted with a new QUIC packet.

In this section, we specify a simple response algorithm when a spurious loss is detected by acknowledgements. Implementations would need to carefully evaluate the impact of using this algorithm in different environments that may experience sudden change in available capacity (e.g., due to variable radio capacity, a routing change, or a mobility event).

When a packet loss is detected via acknowledgements, a CUBIC implementation MAY save the current value of the following variables before the congestion window is reduced.

\[
\begin{align*}
\text{prior\_cwnd} &= \text{cwnd} \\
\text{prior\_ssthresh} &= \text{ssthresh} \\
\text{prior\_W} &= \text{W} \\
\text{max} &= \text{max} \\
\text{prior\_K} &= \text{K} \\
\text{prior\_epoch} &= \text{epoch} \\
\text{start} &= \text{start} \\
\text{prior\_W\_est} &= \text{W} \\
\text{est} &= \text{est}
\end{align*}
\]

Once the previously declared packet loss is confirmed to be spurious, CUBIC MAY restore the original values of the above-mentioned variables as follows if the current cwnd is lower than prior\_cwnd. Restoring the original values ensures that CUBIC’s performance is similar to what it would be without spurious losses.
\[
cwnd = \text{prior}_cwnd \\
sssthresh = \text{prior}_sssthresh \\
W = \text{prior}_W \\
\text{max} \\
K = \text{prior}_K \\
\text{epoch} = \text{prior}_\text{epoch} \\
\text{start} \\
W = \text{prior}_W \\
\text{est} \\
\text{est}
\]

In rare cases, when the detection happens long after a spurious loss event and the current \textunderscore cwnd\_ is already higher than \textunderscore prior\_cwnd\_, CUBIC SHOULD continue to use the current and the most recent values of these variables.

4.10. Slow Start

CUBIC MUST employ a slow-start algorithm, when \_cwnd\_ is no more than \_sssthresh\_. In general, CUBIC SHOULD use the \text{HyStart++} slow start algorithm [I-D.ietf-tcpm-hystartplusplus], or MAY use the Reno TCP slow start algorithm [RFC5681] in the rare cases when \text{HyStart++} is not suitable. Experimental alternatives include hybrid slow start [HR11], a predecessor to \text{HyStart++} that some CUBIC implementations have used as the default for the last decade, and limited slow start [RFC3742]. Whichever start-up algorithm is used, work might be needed to ensure that the end of slow start and the first multiplicative decrease of congestion avoidance work well together.

When CUBIC uses \text{HyStart++} [I-D.ietf-tcpm-hystartplusplus], it may exit the first slow start without incurring any packet loss and thus \_W\_max\_ is undefined. In this special case, CUBIC switches to congestion avoidance and increases its congestion window size using Figure 1, where \_t\_ is the elapsed time since the beginning of the current congestion avoidance, \_K\_ is set to 0, and \_W\_max\_ is set to the congestion window size at the beginning of the current congestion avoidance stage.

5. Discussion

In this section, we further discuss the safety features of CUBIC following the guidelines specified in [RFC5033].
With a deterministic loss model where the number of packets between two successive packet losses is always \( \frac{1}{p} \), CUBIC always operates with the concave window profile, which greatly simplifies the performance analysis of CUBIC. The average window size of CUBIC can be obtained by the following function:

\[
\text{AVG}_W = \frac{4}{C} \cdot \frac{\left(3 + \frac{4}{3}\right)}{\text{cubic}} \cdot \frac{4}{3} \cdot \frac{\text{RTT}}{} \cdot \frac{4}{\left(1 - \frac{4}{3}\right) \cdot \text{cubic}} \cdot \frac{4}{3} \cdot \frac{\left(1 - \frac{1}{p}\right)}{} \cdot \frac{\text{RTT}}{}
\]

Figure 6

With \( \text{cubic} \) set to 0.7, the above formula reduces to:

\[
\text{AVG}_W = \frac{4}{C} \cdot \frac{3.7}{\text{cubic}} \cdot \frac{4}{3} \cdot \frac{\left(1 - \frac{1}{p}\right)}{} \cdot \frac{\text{RTT}}{}
\]

Figure 7

We will determine the value of \( C \) in the following subsection using Figure 7.

5.1. Fairness to Reno

In environments where Reno is able to make reasonable use of the available bandwidth, CUBIC does not significantly change this state.

Reno performs well in the following two types of networks:

1. networks with a small bandwidth-delay product (BDP)
2. networks with a short RTTs, but not necessarily a small BDP

CUBIC is designed to behave very similarly to Reno in the above two types of networks. The following two tables show the average window sizes of Reno TCP, HSTCP, and CUBIC TCP. The average window sizes of Reno TCP and HSTCP are from [RFC3649]. The average window size of CUBIC is calculated using Figure 7 and the CUBIC Reno-friendly region for three different values of \( C \).
### Table 1: Reno TCP, HSTCP, and CUBIC with RTT = 0.1 seconds

Table 1 describes the response function of Reno TCP, HSTCP, and CUBIC in networks with _RTT_ = 0.1 seconds. The average window size is in MSS-sized segments.

<table>
<thead>
<tr>
<th>Loss Rate P</th>
<th>Reno</th>
<th>HSTCP</th>
<th>CUBIC (C=0.04)</th>
<th>CUBIC (C=0.4)</th>
<th>CUBIC (C=4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0e-02</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>1.0e-03</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>59</td>
</tr>
<tr>
<td>1.0e-04</td>
<td>120</td>
<td>263</td>
<td>120</td>
<td>187</td>
<td>333</td>
</tr>
<tr>
<td>1.0e-05</td>
<td>379</td>
<td>1795</td>
<td>593</td>
<td>1054</td>
<td>1874</td>
</tr>
<tr>
<td>1.0e-06</td>
<td>1200</td>
<td>12280</td>
<td>3332</td>
<td>5926</td>
<td>10538</td>
</tr>
<tr>
<td>1.0e-07</td>
<td>3795</td>
<td>83981</td>
<td>18740</td>
<td>33325</td>
<td>59261</td>
</tr>
<tr>
<td>1.0e-08</td>
<td>12000</td>
<td>574356</td>
<td>105383</td>
<td>187400</td>
<td>333250</td>
</tr>
</tbody>
</table>

Table 2: Reno TCP, HSTCP, and CUBIC with RTT = 0.01 seconds

Table 2 describes the response function of Reno TCP, HSTCP, and CUBIC in networks with _RTT_ = 0.01 seconds. The average window size is in MSS-sized segments.

<table>
<thead>
<tr>
<th>Loss Rate P</th>
<th>Reno</th>
<th>HSTCP</th>
<th>CUBIC (C=0.04)</th>
<th>CUBIC (C=0.4)</th>
<th>CUBIC (C=4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0e-02</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>1.0e-03</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
<td>38</td>
</tr>
<tr>
<td>1.0e-04</td>
<td>120</td>
<td>263</td>
<td>120</td>
<td>120</td>
<td>120</td>
</tr>
<tr>
<td>1.0e-05</td>
<td>379</td>
<td>1795</td>
<td>379</td>
<td>379</td>
<td>379</td>
</tr>
<tr>
<td>1.0e-06</td>
<td>1200</td>
<td>12280</td>
<td>1200</td>
<td>1200</td>
<td>1874</td>
</tr>
<tr>
<td>1.0e-07</td>
<td>3795</td>
<td>83981</td>
<td>3795</td>
<td>5926</td>
<td>10538</td>
</tr>
<tr>
<td>1.0e-08</td>
<td>12000</td>
<td>574356</td>
<td>18740</td>
<td>33325</td>
<td>59261</td>
</tr>
</tbody>
</table>
Table 2 describes the response function of Reno TCP, HSTCP, and CUBIC in networks with \_RTT\_ = 0.01 seconds. The average window size is in MSS-sized segments.

Both tables show that CUBIC with any of these three \_C\_ values is more friendly to Reno TCP than HSTCP, especially in networks with a short \_RTT\_ where Reno TCP performs reasonably well. For example, in a network with \_RTT\_ = 0.01 seconds and p=10^-6, Reno TCP has an average window of 1200 packets. If the packet size is 1500 bytes, then Reno TCP can achieve an average rate of 1.44 Gbps. In this case, CUBIC with \_C\_=0.04 or \_C\_=0.4 achieves exactly the same rate as Reno TCP, whereas HSTCP is about ten times more aggressive than Reno TCP.

We can see that \_C\_ determines the aggressiveness of CUBIC in competing with other congestion control algorithms for bandwidth. CUBIC is more friendly to Reno TCP, if the value of \_C\_ is lower. However, we do not recommend setting \_C\_ to a very low value like 0.04, since CUBIC with a low \_C\_ cannot efficiently use the bandwidth in fast and long-distance networks. Based on these observations and extensive deployment experience, we find \_C\_=0.4 gives a good balance between Reno-friendliness and aggressiveness of window increase. Therefore, \_C\_ SHOULD be set to 0.4. With \_C\_ set to 0.4, Figure 7 is reduced to:

\[
\text{AVG}_W = \frac{4}{3} \frac{1}{\text{RTT}} \frac{1}{\text{cubic}} \times 4 \frac{1}{3} \frac{1}{p}
\]

Figure 8

Figure 8 is then used in the next subsection to show the scalability of CUBIC.

5.2. Using Spare Capacity

CUBIC uses a more aggressive window increase function than Reno for fast and long-distance networks.

The following table shows that to achieve the 10 Gbps rate, Reno TCP requires a packet loss rate of 2.0e-10, while CUBIC TCP requires a packet loss rate of 2.9e-8.
Table 3: Required packet loss rate for Reno TCP, HSTCP, and CUBIC to achieve a certain throughput

Table 3 describes the required packet loss rate for Reno TCP, HSTCP, and CUBIC to achieve a certain throughput. We use 1500-byte packets and an _RTT_ of 0.1 seconds.

Our test results in [HLRX07] indicate that CUBIC uses the spare bandwidth left unused by existing Reno TCP flows in the same bottleneck link without taking away much bandwidth from the existing flows.

5.3. Difficult Environments

CUBIC is designed to remedy the poor performance of Reno in fast and long-distance networks.

5.4. Investigating a Range of Environments

CUBIC has been extensively studied using simulations, testbed emulations, Internet experiments, and Internet measurements, covering a wide range of network environments [HLRX07][H16][CEHRX09][HR11][BSCLU13][LBEWK16]. They have convincingly demonstrated that CUBIC delivers substantial benefits over classical Reno congestion control [RFC5681].

Same as Reno, CUBIC is a loss-based congestion control algorithm. Because CUBIC is designed to be more aggressive (due to a faster window increase function and bigger multiplicative decrease factor) than Reno in fast and long-distance networks, it can fill large drop-tail buffers more quickly than Reno and increases the risk of a standing queue [RFC8511]. In this case, proper queue sizing and management [RFC7567] could be used to mitigate the risk to some extent and reduce the packet queuing delay. Also, in large-BDP
networks after a congestion event, CUBIC, due its cubic window increase function, recovers quickly to the highest link utilization point. This means that link utilization is less sensitive to an active queue management (AQM) target that is lower than the amplitude of the whole sawtooth.

Similar to Reno, the performance of CUBIC as a loss-based congestion control algorithm suffers in networks where a packet loss is not a good indication of bandwidth utilization, such as wireless or mobile networks [LIU16].

5.5. Protection against Congestion Collapse

With regard to the potential of causing congestion collapse, CUBIC behaves like Reno, since CUBIC modifies only the window adjustment algorithm of Reno. Thus, it does not modify the ACK clocking and timeout behaviors of Reno.

CUBIC also satisfies the "full backoff" requirement as described in [RFC5033]. After reducing the sending rate to one packet per RTT in response to congestion events due to ECN-Echo ACKs, CUBIC then exponentially increases the transmission timer for each packet retransmission while congestion persists.

5.6. Fairness within the Alternative Congestion Control Algorithm

CUBIC ensures convergence of competing CUBIC flows with the same RTT in the same bottleneck links to an equal throughput. When competing flows have different RTT values, their throughput ratio is linearly proportional to the inverse of their RTT ratios. This is true independently of the level of statistical multiplexing on the link. The convergence time depends on the network environments (e.g., bandwidth, RTT) and the level of statistical multiplexing, as mentioned in Section 3.4.

5.7. Performance with Misbehaving Nodes and Outside Attackers

This is not considered in the current CUBIC design.

5.8. Behavior for Application-Limited Flows

A flow is application-limited if it is currently sending less than what is allowed by the congestion window. This can happen if the flow is limited by either the sender application or the receiver application (via the receiver advertised window) and thus sends less data than what is allowed by the sender’s congestion window.
CUBIC does not increase its congestion window if a flow is application-limited. Section 4.2 requires that \( t \) in Figure 1 does not include application-limited periods, such as idle periods, otherwise \( W_{cubic}(t) \) might be very high after restarting from these periods.

5.9. Responses to Sudden or Transient Events

If there is a sudden increase in capacity, e.g., due to variable radio capacity, a routing change, or a mobility event, CUBIC is designed to utilize the newly available capacity faster than Reno.

On the other hand, if there is a sudden decrease in capacity, CUBIC reduces more slowly than Reno. This remains true whether or not CUBIC is in Reno-friendly mode and whether or not fast convergence is enabled.

5.10. Incremental Deployment

CUBIC requires only changes to the congestion control at the sender, and it does not require any changes at receivers. That is, a CUBIC sender works correctly with Reno receivers. In addition, CUBIC does not require any changes to routers and does not require any assistance from routers.

6. Security Considerations

CUBIC makes no changes to the underlying security of TCP. More information about TCP security concerns can be found in [RFC5681].

7. IANA Considerations

This document does not require any IANA actions.

8. References

8.1. Normative References

[I-D.ietf-tcpm-hystartplusplus]


8.2. Informative References


Appendix A. Acknowledgments

Richard Scheffenegger and Alexander Zimmermann originally co-authored [RFC8312].

These individuals suggested improvements to this document:

* Bob Briscoe
* Christian Huitema
* Gorry Fairhurst
* Jonathan Morton
* Juhamatti Kuusisaari
* Junho Choi
* Markku Kojo
* Martin Thomson
* Matt Mathis
* Matt Olson
* Michael Welzl
* Mirja Kuehlewind
* Mohit P. Tahiliani
* Neal Cardwell
* Praveen Balasubramanian
Appendix B. Evolution of CUBIC

B.1. Since draft-ietf-tcpm-rfc8312bis-06

* RFC7661 is safe even when cwnd grows beyond rwnd (#143
  (https://github.com/NTAP/rfc8312bis/issues/143))

B.2. Since draft-ietf-tcpm-rfc8312bis-05

* Clarify meaning of "application-limited" in Section 5.8 (#137
  (https://github.com/NTAP/rfc8312bis/issues/137))

* Create new subsections for spurious timeouts and spurious loss via
  ACK (#90 (https://github.com/NTAP/rfc8312bis/issues/90))

* Brief discussion of convergence in Section 5.6 (#96
  (https://github.com/NTAP/rfc8312bis/issues/96))

* Add more test results to Section 5 and update some references (#91
  (https://github.com/NTAP/rfc8312bis/issues/91))

* Change wording around setting ssthresh (#131
  (https://github.com/NTAP/rfc8312bis/issues/131))

B.3. Since draft-ietf-tcpm-rfc8312bis-04

* Fix incorrect math (#106 (https://github.com/NTAP/rfc8312bis/
  issues/106))

* Update RFC5681 (#99 (https://github.com/NTAP/rfc8312bis/
  issues/99))

* Clarify what we mean by "new ACK" and use it in the text in more places. (#101 (https://github.com/NTAP/rfc8312bis/issues/101))

* Rewrite the Responses to Sudden or Transient Events section (#98 (https://github.com/NTAP/rfc8312bis/issues/98))

* Remove confusing text about _cwnd_start_ in Section 4.2 (#100 (https://github.com/NTAP/rfc8312bis/issues/100))

* Change terminology from "AIMD" to "Reno" (#108 (https://github.com/NTAP/rfc8312bis/issues/108))

* Moved MUST NOT from app-limited section to main cubic AI section (#97 (https://github.com/NTAP/rfc8312bis/issues/97))

* Clarify cwnd decrease during multiplicative decrease (#102 (https://github.com/NTAP/rfc8312bis/issues/102))

* Clarify text around queuing and slow adaptation of CUBIC in wireless environments (#94 (https://github.com/NTAP/rfc8312bis/issues/94))

* Set lower bound of cwnd to 1 MSS and use retransmit timer thereafter (#83 (https://github.com/NTAP/rfc8312bis/issues/83))

* Use FlightSize instead of cwnd to update ssthresh (#114 (https://github.com/NTAP/rfc8312bis/issues/114))

B.4. Since draft-ietf-tcpm-rfc8312bis-03

* Remove reference from abstract (#82 (https://github.com/NTAP/rfc8312bis/pull/82))

B.5. Since draft-ietf-tcpm-rfc8312bis-02

* Description of packet loss rate _p_ (#65 (https://github.com/NTAP/rfc8312bis/issues/65))

* Clarification of TCP Friendly Equation for ABC and Delayed ACK (#66 (https://github.com/NTAP/rfc8312bis/issues/66))

* add applicability to QUIC and SCTP (#61 (https://github.com/NTAP/rfc8312bis/issues/61))
* clarity on setting alpha__aimd_ to 1 (#68 (https://github.com/NTAP/rfc8312bis/issues/68))

* introduce alpha__cubic_ (#64 (https://github.com/NTAP/rfc8312bis/issues/64))

* clarify _cwnd_ growth in convex region (#69 (https://github.com/NTAP/rfc8312bis/issues/69))

* add guidance for using bytes and mention that segments count is decimal (#67 (https://github.com/NTAP/rfc8312bis/issues/67))

* add loss events detected by RACK and QUIC loss detection (#62 (https://github.com/NTAP/rfc8312bis/issues/62))

B.6. Since draft-ietf-tcpm-rfc8312bis-01

* address Michael Scharf’s editorial suggestions. (#59 (https://github.com/NTAP/rfc8312bis/issues/59))

* add "Note to the RFC Editor" about removing underscores

B.7. Since draft-ietf-tcpm-rfc8312bis-00

* use updated xml2rfc with better text rendering of subscripts

B.8. Since draft-eggert-tcpm-rfc8312bis-03

* fix spelling nits

* rename to draft-ietf

* define _W_max_ more clearly

B.9. Since draft-eggert-tcpm-rfc8312bis-02

* add definition for segments_acked and alpha__aimd_. (#47 (https://github.com/NTAP/rfc8312bis/issues/47))

* fix a mistake in _W_max_ calculation in the fast convergence section. (#51 (https://github.com/NTAP/rfc8312bis/issues/51))

* clarity on setting _ssthresh_ and _cwnd_start_ during multiplicative decrease. (#53 (https://github.com/NTAP/rfc8312bis/issues/53))

B.10. Since draft-eggert-tcpm-rfc8312bis-01
* rename TCP-Friendly to AIMD-Friendly and rename Standard TCP to AIMD TCP to avoid confusion as CUBIC has been widely used on the Internet. (#38 (https://github.com/NTAP/rfc8312bis/issues/38))

* change introductory text to reflect the significant broader deployment of CUBIC on the Internet. (#39 (https://github.com/NTAP/rfc8312bis/issues/39))

* rephrase introduction to avoid referring to variables that have not been defined yet.

B.11. Since draft-eggert-tcpm-rfc8312bis-00

* acknowledge former co-authors (#15 (https://github.com/NTAP/rfc8312bis/issues/15))

* prevent cwnd from becoming less than two (#7 (https://github.com/NTAP/rfc8312bis/issues/7))

* add list of variables and constants (#5 (https://github.com/NTAP/rfc8312bis/issues/5), #6 (https://github.com/NTAP/rfc8312bis/issues/6))


* update W_est to use AIMD approach (#20 (https://github.com/NTAP/rfc8312bis/issues/20))

* set alpha_aimd to 1 once W_est reaches W_max (#2 (https://github.com/NTAP/rfc8312bis/issues/2))

* add Vidhi as co-author (#17 (https://github.com/NTAP/rfc8312bis/issues/17))

* note for Fast Recovery during cwnd decrease due to congestion event (#11 (https://github.com/NTAP/rfc8312bis/issues/11))

* add section for spurious congestion events (#23 (https://github.com/NTAP/rfc8312bis/issues/23))

* initialize W_est after timeout and remove variable _W_(last_max) (#28 (https://github.com/NTAP/rfc8312bis/issues/28))

B.12. Since RFC8312

B.13. Since the Original Paper

CUBIC has gone through a few changes since the initial release [HRX08] of its algorithm and implementation. Below we highlight the differences between its original paper and [RFC8312].

* The original paper [HRX08] includes the pseudocode of CUBIC implementation using Linux’s pluggable congestion control framework, which excludes system-specific optimizations. The simplified pseudocode might be a good source to start with and understand CUBIC.

* [HRX08] also includes experimental results showing its performance and fairness.

* The definition of beta_cubic constant was changed in [RFC8312]. For example, beta_cubic in the original paper was the window decrease constant while [RFC8312] changed it to CUBIC multiplication decrease factor. With this change, the current congestion window size after a congestion event in [RFC8312] was beta_cubic * W_max while it was (1-beta_cubic) * W_max in the original paper.

* Its pseudocode used W_last_max while [RFC8312] used W_max.

* Its AIMD-friendly window was W_tcp while [RFC8312] used W_est.

Authors’ Addresses

Lisong Xu
University of Nebraska-Lincoln
Department of Computer Science and Engineering
Lincoln, NE 68588-0115
United States of America
Email: xu@unl.edu
URI: https://cse.unl.edu/~xu/
TCPLS: Modern Transport Services with TCP and TLS

draft-piraux-tcpls-01

Abstract

This document specifies a protocol leveraging TCP and TLS to provide modern transport services such as multiplexing, connection migration and multipath in a secure manner.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/mpiraux/draft-piraux-tcpls.

Status of This Memo

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

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

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

This Internet-Draft will expire on 8 September 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction .................................................. 3
2. Conventions and Definitions .................................. 5
   2.1. Notational conventions .................................. 5
3. Modern Transport Services ...................................... 6
4. TCPLS Overview ................................................ 6
   4.1. Multiple Streams ......................................... 7
   4.2. Multiple TCP connections ................................. 8
       4.2.1. Joining TCP connections ............................ 8
       4.2.2. Failover ........................................... 9
       4.2.3. Migration .......................................... 10
       4.2.4. Multipath .......................................... 10
   4.3. Record protection ........................................ 10
   4.4. Closing a TCPLS session .................................. 11
5. TCPLS Protocol ................................................ 11
   5.1. TCPLS TLS Extensions .................................... 12
       5.1.1. TCPLS ............................................... 12
       5.1.2. TCPLS Join ......................................... 12
   5.2. TCPLS Frames .............................................. 13
       5.2.1. Padding frame ....................................... 13
       5.2.2. Ping frame ......................................... 14
       5.2.3. Stream frame ....................................... 14
       5.2.4. ACK frame ......................................... 15
       5.2.5. New Token frame .................................... 15
       5.2.6. Connection Reset frame .............................. 16
       5.2.7. New Address frame .................................. 16
       5.2.8. Remove Address frame ................................ 17
6. Security Considerations ....................................... 17
7. IANA Considerations .......................................... 17
   7.1. TCPLS TLS Extensions .................................... 17
   7.2. TCPLS Frames .............................................. 18
8. References .................................................... 18
   8.1. Normative References .................................... 18
   8.2. Informative References .................................. 18
Acknowledgments .................................................. 20
Change log ....................................................... 20
Since draft-piraux-tcpls-00 .................................... 20
Authors’ Addresses .............................................. 20
1. Introduction

The TCP/IP protocol stack continuously evolves. In the early days, most applications were interacting with the transport layer (mainly TCP, but also UDP) using the socket API. This is illustrated in Figure 1.

```
+------------------------------+
|          Application         |
+------------------------------+
|            TCP/UDP           |
+------------------------------+
|             IPv4             |
```

Figure 1: The classical TCP/IP protocol stack

The TCP/IP stack has slowly evolved and the figure above does not anymore describe current Internet applications. IPv6 is now widely deployed next to IPv4 in the network layer. In the transport layer, protocols such as SCTP [RFC4960] or DCCP [RFC6335] and TCP extensions including Multipath TCP [RFC8684] or tcpcrypt [RFC8548] have been specified. The security aspects of the TCP/IP protocol suite are much more important today than in the past [RFC7258]. Many applications rely on TLS [RFC8446] and their stack is similar to the one shown in Figure 2.

```
+------------------------------+
|          Application         |
+------------------------------+
|             TLS              |
+------------------------------+
|             TCP              |
+------------------------------+
|          IPv4/IPv6           |
```

Figure 2: Today’s TCP/IP protocol stack

Recently, the IETF went one step further in improving the transport layer with the QUIC protocol [RFC9000]. QUIC is a new secure transport protocol primarily designed for HTTP/3. It includes the reliability and congestion control features that are part of TCP and integrates the security features of TLS 1.3 [RFC8446]. This close integration between the reliability and security features brings a lot of benefits in QUIC. QUIC runs above UDP to be able to pass through most middleboxes and to be implementable in user space. While QUIC reuses TLS, it does not strictly layer TLS on top of UDP.
as DTLS [I-D.ietf-tls-dtls13]. This organization, illustrated in Figure 3 provides much more flexibility than simply layering TLS above UDP. For example, the QUIC migration capabilities enable an application to migrate an existing QUIC session from an IPv4 path to an IPv6 one.

![Figure 3: QUIC protocol stack](image)

In this document, we revisit how TCP and TLS 1.3 can be used to provide modern transport services to applications. We apply a similar principle and combine TCP and TLS 1.3 in a protocol that we call TCPLS. TCPLS leverages the security features of TLS 1.3 like QUIC, but without begin simply layered above a single TCP connection. In addition, TCPLS reuses the existing TCP stacks and TCP’s wider support in current networks. A preliminary version of the TCPLS protocol is described in [CONEXT21].

![Figure 4: TCPLS in the TCP/IP protocol stack](image)

In this document, we use the term TLS/TCP to refer to the TLS 1.3 protocol running over one TCP connection. We reserve the word TCPLS for the protocol proposed in this document.

This document is organized as follows. First, Section 3 summarizes the different types of services that modern transports expose to application. Section 4 gives an overview of TCPLS and how it supports these services. Finally, Section 5 describes the TCPLS in more details and the TLS Extensions introduced in this document.
2. Conventions and Definitions

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. Notational conventions

This document uses the same conventions as defined in Section 1.3 of [RFC9000].

This document uses network byte order (that is, big endian) values. Fields are placed starting from the high-order bits of each byte.

3. Modern Transport Services

Application requirements and the devices they run on evolve over time. In the early days, most applications involved single-file transfer and ran on single-homed computers with a fixed-line network. Today, web-based applications require exchanging multiple objects, with different priorities, on devices that can move from one access network to another and that often have multiple access networks available. Security is also a key requirement of applications that evolved from only guaranteeing the confidentiality and integrity of application messages to also preventing pervasive monitoring.

With TCP and TLS/TCP, applications use a single connection that supports a single bytestream in each direction. Some TCP applications such as HTTP/2 [RFC7540] use multiple streams, but these are mapped to a single TCP connection which leads to Head-of-Line (HoL) blocking when packet losses occur. SCTP [RFC4960] supports multiple truly-concurrent streams and QUIC adopted a similar approach to prevent HoL blocking.

Modern transport services also changed the utilization of the underlying network. With TCP, when a host creates a connection, it is bound to the IP addresses used by the client and the server during the handshake. When the client moves and receives a different IP address, it has to reestablish all TCP connections bound to the previous address. When the client and the server are dual-stack, they cannot easily switch from one address family to another. Happy Eyeballs [RFC8305] provides a partial answer to this problem for web applications with heuristics that clients can use to probe TCP connections with different address families. With Multipath TCP, the client and the server can learn other addresses of the remote host and combine several TCP connections within a single Multipath TCP
connection that is exposed to the application. This supports various use cases [RFC8041]. QUIC [RFC9000] enables applications to migrate from one network path to another, but not to simultaneously use different paths.

4. TCPLS Overview

In order for TCPLS to be widely compatible with middleboxes that inspect TCP segments and TLS records, TCPLS does not modify the TCP connection establishment and only adds a TLS extension to the TLS handshake. Figure 5 illustrates the opening of a TCPLS session which starts with the TCP 3-way handshake, followed by the TLS handshake. In the Extensions of the ClientHello and in the server EncryptedExtensions, the tcpls TLS Extension is introduced to announce the support of TCPLS.

```
Client                                   Server
<table>
<thead>
<tr>
<th>SYN</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN+ACK</td>
</tr>
<tr>
<td>&lt;------------------------------------------</td>
</tr>
<tr>
<td>ACK, TLS ClientHello + tcpls</td>
</tr>
<tr>
<td>------------------------------------------</td>
</tr>
<tr>
<td>TLS ServerHello, TLS EncryptedExtensions</td>
</tr>
<tr>
<td>+ tcpls, ...</td>
</tr>
<tr>
<td>&lt;------------------------------------------</td>
</tr>
<tr>
<td>TLS Finished</td>
</tr>
<tr>
<td>------------------------------------------</td>
</tr>
</tbody>
</table>
```

**Figure 5: Starting a TCPLS session**

TCP/TLS offers a single encrypted bytestream service to the application. To achieve this, TLS records are used to encrypt and secure chunks of the application bytestream and are then sent through the TCP bytestream. TCPLS leverages TLS records differently. TCPLS defines its own framing mechanism that allows encoding both application data and control information. A TCPLS frame is the basic unit of information for TCPLS. One or more TCPLS frames can be placed inside a TLS record. A TCPLS frame always fits in a single record. This TLS record is then reliably transported by a TCP connection. Figure 6 illustrates the relationship between TCPLS frames and TLS records.
Figure 6: The first TLS record contains three TCPLS frames

4.1. Multiple Streams

TCPLS extends the service provided by TCP with streams. Streams are independent bidirectional byte streams that can be used by applications to concurrently convey several objects over a TCPLS session. Streams can be opened by the client and by the server.

Streams are identified by a 32-bit unsigned integer. The parity of this number indicates the initiator of the stream. The client opens even-numbered streams while the server opens odd-numbered streams. Streams are opened in sequence, e.g. a client that has opened stream 0 will use stream 2 as the next one.

Data is exchanged using Stream frames whose format is described in Section 5.2.3. Each Stream frame carries a chunk of data of a given stream. Applications can mark the end of a stream to close it.

Similarly to HTTP/2 [RFC7540], conveying several streams on a single TCP connection introduces Head-of-Line (HoL) blocking between the streams. To alleviate this, TCPLS provides means to the application to choose the degree of HoL blocking resilience it needs for its application objects by spreading streams among different underlying TCP connections.
4.2. Multiple TCP connections

TCPLS is not restricted to using a single TCP connection to exchange frames. A TCPLS session starts with the TCP connection that was used to transport the TLS handshake. After this handshake, other TCP connections can be added to a TCPLS session, either to spread the load or for failover. TCPLS manages the utilization of the underlying TCP connections within a TCPLS session.

Multipath TCP enables both the client and the server to establish additional TCP connections. However, experience has shown that additional subflows are only established by the clients. TCPLS focuses on this deployment and only allows clients to create additional TCP connections.

Using Multipath TCP, a client can try establishing a new TCP connection at any time. If a server wishes to restrict the number of TCP connections that correspond to one Multipath TCP connection, it has to respond with RST to the in excess connection attempts.

TCPLS takes another approach. To control the number of connections that a client can establish, a TCPLS server supplies unique tokens. A client includes one of the server supplied tokens when it attaches a new TCP connection to a TCPLS session. Each token can only be used once, hence limiting the amount of additional TCP connections.

TCPLS endpoints can advertise their local addresses, allowing new TCP connections for a given TCPLS session to be established between new pairs of addresses. When an endpoint is no more willing new TCP connections to use one of its advertised addresses, it can remove this address from the TCPLS session.

4.2.1. Joining TCP connections

The TCPLS server provides tokens to the client in order to join new TCP connections to the TCPLS session. Figure 7 illustrates a client and server first establishing a new TCPLS session as described in Section 4. Then the server sends a token over this connection using the New Token frame. Each token has a sequence number (e.g. 1) and a value (e.g. "abc"). The client uses this token to open a new TCP connection and initiates the TCPLS handshake. It adds the token inside the TCPLS Join TLS extension in the ClientHello.
When receiving a TCPLS Join Extension, the server validates the token and associates the TCP connection to the TCPLS session.

Each TCP connection that is part of a TCPLS session is identified by a 32-bit unsigned integer called its Connection ID. The first TCP connection of a session corresponds to Connection ID 0. When joining a new connection, the sequence number of the token, i.e. 1 in our example, becomes the Connection ID of the connection. The Connection ID enables the Client and the Server to identify a specific TCP connection within a given TCPLS session.

4.2.2. Failover

TCPLS supports two types of failover. In make-before-break, the client creates a TCP connection using the procedure described in Section 4.2.1 but only uses it once the initial connection fails.

In break-before-make, the client creates the initial TCP connection and uses it for the TCPLS handshake and the data. The server advertises one or more tokens over this connection. Upon failure of the initial TCP connection, the client initiates a second TCP connection using the server-provided token.

In both cases, some records sent by the client or the server might be in transit when the failure occurs. Some of these records could have been partially received but not yet delivered to the TCPLS layer when the underlying TCP connection fails. Other records could have already been received, decrypted and data of their frames could have been delivered to the application. To prevent data losses and duplication, TCPLS includes its own acknowledgments.
A TCPLS receiver acknowledges the received records using the ACK frame. Records are acknowledged after the record protection has been successfully removed. This enables the sender to know which records have been received. TCPLS enables the endpoint to send acknowledgments for a TCP connection over any connections, e.g. not only the receiving connection.

4.2.3. Migration

To migrate from a given TCP connection, an endpoint stops transmitting over this TCP connection and sends the following frames on other TCP connections. It leverages the acknowledgments to retransmit the frames of TLS records that have not been yet acknowledged.

When an endpoint abortfully closes a TCP connection, its peer leverages the acknowledgments to retransmit the TLS records that were not acknowledged.

4.2.4. Multipath

TCPLS also supports the utilization of different TCP connections, over different paths or interfaces, to improve throughput or spread stream frames over different TCP connections. When the endpoints have opened several TCP connections, they can send frames over the connections. TCPLS can send all the stream frames belonging to a given stream over one or more underlying TCP connections. The latter enables bandwidth aggregation by using TCP connections established over different network paths.

4.3. Record protection

When adding new TCP connections to a TCPLS session, an endpoint does not complete the TLS handshake. TCPLS provides a nonce construction for TLS record protection that is used for all connections of a session. This reduces the cryptographic cost of adding connections. The endpoints SHOULD send TLS messages to form an apparent complete TLS handshake to middleboxes.

In order to use the TLS session over multiple connections, TCPLS adds a record sequence number space per connection that is maintained independently at both sides. Each record sent over a TCPLS session is identified by the Connection ID of its connection and its record sequence number. Each record nonce is constructed as defined in Figure 8.
This construction guarantees that every TLS record sent over the TLS session is protected with a unique nonce. As in TLS 1.3, the per-connection record sequence is implicit.

### 4.4. Closing a TCPLS session

Endpoints notify their peers that they do not intend to send more data over a given TCPLS session by sending a TLS Alert "close_notify". The alert can be sent over one or more TCP connections of the session. The alert MUST be sent before closing the last TCP connection of the TCPLS session. The endpoint MAY close its side of the TCP connections after sending the alert.

When all TCP connections of a session are closed and the TLS Alert "close_notify" was exchanged in both directions, the TCPLS session is considered as closed.

We leave defining an abortful and idle session closure mechanisms for future versions of this document.

### 5. TCPLS Protocol

#### 5.1. TCPLS TLS Extensions

This document specifies two TLS extensions used by TCPLS. The first, "tcpls", is used to announce the support of TCPLS. The second, "tcpls_join", is used to join a TCP connection to a TCPLS session. Their types are defined as follows.

```c
enum {
    tcpls(TBD1),
    tcpls_join(TBD2),
    (65535)
} ExtensionType;
```
The table below indicates the TLS messages where these extensions can appear. "CH" indicates ClientHello while "EE" indicates EncryptedExtensions.

<table>
<thead>
<tr>
<th>Extension</th>
<th>Allowed TLS messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcpls</td>
<td>CH, EE</td>
</tr>
<tr>
<td>tcpls_join</td>
<td>CH</td>
</tr>
</tbody>
</table>

Table 1: TLS messages allowed to carry TCPLS TLS Extensions

5.1.1. TCPLS

The "tcpls" extension is used by the client and the server to announce their support of TCPLS. The extension contains no value. When it is present in both the ClientHello and the EncryptedExtensions, the endpoints MUST use TCPLS after completing the TLS handshake.

5.1.2. TCPLS Join

```c
struct {
    opaque token<32>;
} Join;
```

The "tcpls_join" extension is used by the client to join the TCP connection on which it is sent to a TCPLS session. The extension contains a Token provided by the server. The client MUST NOT send more than one "tcpls_join" extension in its ClientHello. When receiving a ClientHello with this extension, the server checks that the token is valid and joins the TCP connection to the corresponding TCPLS session. When the token is not valid, the server MUST abort the handshake with an illegal_parameter alert.

By controlling the amount of tokens given to the client, the server can control the number of active TCP connections of a TCPLS session. The server SHOULD replenish the tokens when TCP connections are removed from the TCPLS session.
5.2. TCPLS Frames

TCPLS uses TLS Application Data records to exchange TCPLS frames. After decryption, the record payload consists of a sequence of TCPLS frames. A frame is a Type-Value unit, starting with a byte indicating its frame type followed by type-specific fields. Table 2 lists the frames specified in this document.

<table>
<thead>
<tr>
<th>Type value</th>
<th>Frame name</th>
<th>Rules</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>Padding</td>
<td>N</td>
<td>Section 5.2.1</td>
</tr>
<tr>
<td>0x01</td>
<td>Ping</td>
<td></td>
<td>Section 5.2.2</td>
</tr>
<tr>
<td>0x02-0x03</td>
<td>Stream</td>
<td></td>
<td>Section 5.2.3</td>
</tr>
<tr>
<td>0x04</td>
<td>ACK</td>
<td>N</td>
<td>Section 5.2.4</td>
</tr>
<tr>
<td>0x05</td>
<td>New Token</td>
<td>S</td>
<td>Section 5.2.5</td>
</tr>
<tr>
<td>0x06</td>
<td>Connection Reset</td>
<td></td>
<td>Section 5.2.6</td>
</tr>
<tr>
<td>0x07</td>
<td>New Address</td>
<td></td>
<td>Section 5.2.7</td>
</tr>
<tr>
<td>0x08</td>
<td>Remove Address</td>
<td></td>
<td>Section 5.2.8</td>
</tr>
</tbody>
</table>

Table 2: TCPLS frames

The "Rules" column in Table 2 indicates special requirements regarding certain frames.

N: Non-ack-eliciting. Receiving this frame does not elicit the sending of a TCPLS acknowledgment.

S: Server only. This frame MUST NOT be sent by the client.

5.2.1. Padding frame

This frame has no semantic value. It can be used to mitigate traffic analysis on the TLS records of a TCPLS session. The Padding frame has no content.

Padding frame {
    Type (8) = 0x00,
}

5.2.2. Ping frame

This frame is used to elicit an acknowledgment from its peer. It has no content. When an endpoint receives a Ping frame, it acknowledges the TLS record that contains this frame. This frame can be used by an endpoint to check that its peer can receive TLS records over a particular TCP connection.

\[
\text{Ping frame} \{ \\
\quad \text{Type (8) = 0x01,} \\
\}
\]

5.2.3. Stream frame

This frame is used to carry chunks of data of a given stream.

\[
\text{Stream frame} \{ \\
\quad \text{Type (7) = 0x01,} \\
\quad \text{FIN (1),} \\
\quad \text{Stream ID (32),} \\
\quad \text{Offset (64),} \\
\quad \text{Length (16),} \\
\quad \text{Stream Data (...),} \\
\}
\]

FIN: The last bit of the frame type bit indicates that this Stream frame ends the stream when its value is 1. The last byte of the stream is at the sum of the Offset and Length fields of this frame.

Stream ID: A 32-bit unsigned integer indicating the ID of the stream this frame relates to.

Offset: A 64-bit unsigned integer indicating the offset in bytes of the carried data in the stream.

Length: A 16-bit unsigned integer indicating the length of the Stream Data field.
5.2.4. ACK frame

This frame is sent by the receiver to acknowledge the receipt of TLS records on a particular TCP connection of the TCPLS session. Although the reliability of the data exchange on a connection is handled by TCP, there are situations such as the failure of a TCP connection where a sender does not know whether the TLS frames that it sent have been correctly received by the peer. The ACK frame allows a TCPLS receiver to indicate the highest TLS record sequence number received on a specific connection. The ACK frame can be sent over any TCP connection of a TCPLS session.

ACK frame {
    Type (8) = 0x04,
    Connection ID (32),
    Highest Record Sequence Received (64),
}

Figure 12: ACK frame format

Connection ID: A 32-bit unsigned integer indicating the TCP connection for which the acknowledgment was sent.

Highest Record Sequence Received: A 64-bit unsigned integer indicating the highest TLS record sequence number received on the connection indicated by the Connection ID.

5.2.5. New Token frame

This frame is used by the server to provide tokens to the client. Each token can be used to join a new TCP connection to the TCPLS session, as described in Section 4.2.1. Clients MUST NOT send New Token frames.

New Token frame {
    Type (8) = 0x05,
    Sequence (8),
    Token (256),
}

Figure 13: New Token frame format

Sequence: A 8-bit unsigned integer indicating the sequence number of this token

Token: A 32-byte opaque value that can be used as a token by the client.
5.2.6. Connection Reset frame

This frame is used by the receiver to inform the sender that a TCP connection has been reset.

Connection Reset frame {
    Type (8) = 0x06,
    Connection ID (32)
}

Figure 14: Connection Reset format

Connection ID: A 32-bit unsigned integer indicating the ID of the connection that failed.

5.2.7. New Address frame

This frame is used by an endpoint to add a new local address to the TCPLS session. This address can then be used to establish new TCP connections. The server advertises addresses that the client can use as destination when adding TCP connections. The client advertises address that it can use as source when adding TCP connections.

New Address frame {
    Type (8) = 0x07,
    Address ID (8),
    Address Version (8),
    Address (...),
    Port (16),
}

Figure 15: New Address format

Address ID: A 8-bit identifier for this address. For a given Address ID, an endpoint receiving a frame with a content that differs from previously received frames MUST ignore the frame. An endpoint receiving a frame for an Address ID that was previously removed MUST ignore the frame.

Address Version: A 8-bit value identifying the Internet address version of this address. The number 4 indicates IPv4 while 6 indicates IPv6.

Address: The address value. Its size depends on its version. IPv4 addresses are 32-bit long while IPv6 addresses are 128-bit long.

Port: A 16-bit value indicating the TCP port used with this address.
5.2.8. Remove Address frame

This frame is used by an endpoint to announce that it is not willing to use a given address to establish new TCP connections. After receiving this frame, a client MUST NOT establish new TCP connections to the given address. After receiving this frame, an endpoint MUST close all TCP connections using the given address.

Remove Address frame{
    Type (8) = 0x08,
    Address ID (8),
}

Figure 16: Remove Address format

Address ID: A 8-bit identifier for the address to remove. An endpoint receiving a frame for an address that was nonexistent or already removed MUST ignore the frame.

6. Security Considerations

When issuing tokens to the client as presented in Section 4.2.1, the server SHOULD ensure that their values appear as random to observers and cannot be correlated together for a given TCPLS session.

The security considerations for TLS apply to TCPLS. The next versions of this document will elaborate on other security considerations following the guidelines of [RFC3552].

7. IANA Considerations

IANA is requested to create a new "TCPLS" heading for the new registry described in Section 5.2. New registrations in TCPLS registries follow the "Specification Required" policy of [RFC8126].

7.1. TCPLS TLS Extensions

IANA is requested to add the following entries to the existing "TLS ExtensionType Values" registry.
<table>
<thead>
<tr>
<th>Value</th>
<th>Extension Name</th>
<th>TLS 1.3</th>
<th>Recommended</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBD1</td>
<td>tcpls</td>
<td>CH, EE</td>
<td>N</td>
<td>This document</td>
</tr>
<tr>
<td>TBD2</td>
<td>tcpls_join</td>
<td>CH</td>
<td>N</td>
<td>This document</td>
</tr>
</tbody>
</table>

Table 3

Note that "Recommended" is set to N as these extensions are intended for uses as described in this document.

7.2. TCPLS Frames

IANA is requested to create a new registry "TCPLS Frames Types" under the "TCPLS" heading.

The registry governs an 8-bit space. Entries in this registry must include a "Frame name" field containing a short mnemonic for the frame type. The initial content of the registry is present in Table 2, without the "Rules" column.

8. References

8.1. Normative References


8.2. Informative References

Piraux, et al. Expires 8 September 2022 [Page 18]


Acknowledgments

This work has been partially supported by the ‘‘Programme de recherche d’intérêt général WALINNOV - MQUIC project (convention number 1810018)’’ and European Union through the NGI Pointer programme for the TCPLS project (Horizon 2020 Framework Programme, Grant agreement number 871528). The authors thank Quentin De Coninck and Louis Navarre for their comments on the first version of this draft.

Change log

Since draft-piraux-tcpls-00

* Added the addresses exchange mechanism with New Address and Remove Address frames.

Authors’ Addresses

Maxime Piraux
UCLouvain
Email: maxime.piraux@uclouvain.be

Olivier Bonaventure
UCLouvain
Email: olivier.bonaventure@uclouvain.be

Florentin Rochet
University of Edinburgh
Email: frochet@ed.ac.uk