                                                                 S. Ha
                                                              Colorado
                                                         A. Zimmermann

                                                             L. Eggert
                                                       R. Scheffenegger
                                                                NetApp
                                                         February 2018

                       CUBIC for Fast Long-Distance Networks

Abstract

   CUBIC is an extension to the current TCP standards. It differs from
   the current TCP standards only in the congestion control algorithm on
   the sender side. In particular, it uses a cubic function instead of
   a linear window increase function of the current TCP standards to
   improve scalability and stability under fast and long-distance
   networks. CUBIC and its predecessor algorithm have been adopted as
   defaults by Linux and have been used for many years. This document
   provides a specification of CUBIC to enable third-party
   implementations and to solicit community feedback through
   experimentation on the performance of CUBIC.

Copyright Notice

Table of Contents

1. Introduction

   The low utilization problem of TCP in fast long-distance networks is
   well documented in [K03] and [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). [HKLRX06]
   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 TCP standards and
   their variants, including TCP-RENO [RFC5681], TCP-NewReno [RFC6582]
   [RFC6675], SCTP [RFC4960], and TFRC [RFC5348], which use the same
   linear increase function for window growth, which we refer to
   collectively as "Standard TCP" below.

   CUBIC, originally proposed in [HRX08], is a modification to the
   congestion control algorithm of Standard TCP to remedy this problem.
   This document describes the most recent specification of CUBIC.
   Specifically, CUBIC uses a cubic function instead of a linear window
   increase function of Standard TCP to improve scalability and
   stability under fast and long-distance networks.

   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 has 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 Standard TCP in bandwidth usage than BIC-TCP
   while maintaining the strengths of BIC-TCP such as stability, window
   scalability, and RTT fairness. CUBIC has already replaced BIC-TCP as
   the default TCP congestion control algorithm in Linux and has been
   deployed globally by Linux. Through extensive testing in various
   Internet scenarios, we believe that CUBIC is safe for testing and
   deployment in the global Internet.

   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 TCP-friendly, CUBIC is designed to behave like
      Standard TCP in networks with short RTTs and small bandwidth where
      Standard TCP 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.

   Principle 1: 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 Standard TCP 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 is detected by duplicate
   ACKs or Explicit Congestion Notification-Echo (ECN-Echo) ACKs
   [RFC3168], CUBIC registers the congestion window size where it got
   the congestion event as W_max and performs a multiplicative decrease
   of congestion window. After it 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
   W_max so that the concave window increase continues until the window
   size becomes W_max. 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 [CEHRX07]. This
   is because the window size remains almost constant, forming a plateau
   around W_max where network utilization is deemed highest. Under
   steady state, most window size samples of CUBIC are close to W_max,
   thus promoting high network utilization and stability. Note that
   those congestion control algorithms using only convex functions to
   increase the congestion window size have the maximum increments
   around W_max, and thus introduce a large number of packet bursts
   around the saturation point of the network, likely causing frequent
   global loss synchronizations.

Principle 2: CUBIC promotes per-flow fairness to Standard TCP. Note
that Standard TCP performs well under short RTT and small bandwidth
(or small BDP) networks. There is only a scalability problem in
networks with long RTTs and large bandwidth (or large BDP). An
alternative congestion control algorithm to Standard TCP designed to
be friendly to Standard TCP on a per-flow basis must operate to
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 increases its congestion window less
aggressively than Standard TCP, CUBIC simply follows the window size
of Standard TCP to ensure that CUBIC achieves at least the same
throughput as Standard TCP in small BDP networks. We call this
region where CUBIC behaves like Standard TCP, the "TCP-friendly
region".

Principle 3: Two CUBIC flows with different RTTs have their
throughput ratio 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 of the
TCP-friendly region, and thus flows with different RTTs have similar
congestion window sizes under steady state when they operate outside
the TCP-friendly region. This notion of a linear throughput ratio is
similar to that of Standard TCP under high statistical multiplexing
environments where packet losses are independent of individual flow
rates. However, under low statistical multiplexing environments, the
throughput ratio of Standard TCP flows with different RTTs is
quadratically proportional to the inverse of their RTT ratio [XHR04].
CUBIC always ensures the linear throughput ratio independent of the
levels of statistical multiplexing. This is an improvement over
Standard TCP. While there is no consensus on particular throughput
ratios of different RTT flows, we believe that under wired Internet,
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 Standard TCP under low statistical multiplexing
environments).

Principle 4: To balance between the scalability and convergence
speed, CUBIC sets the multiplicative window decrease factor to 0.7
while Standard TCP uses 0.5. While this improves the scalability of
CUBIC, a side effect of this decision is slower convergence,
especially under low statistical multiplexing environments. This
design choice is following the observation that the author of

HighSpeed TCP (HSTCP) [RFC3649] has made along with other researchers
(e.g., [GV02]): the current Internet becomes more asynchronous with
less frequent loss synchronizations with high statistical
multiplexing. Under this environment, 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 under the environments with sufficient statistical
multiplexing, the convergence speed of CUBIC flows is reasonable.

4. CUBIC Congestion Control

   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.
   Let cwnd denote the congestion window size of a flow, and ssthresh
   denote the slow-start threshold.

4.1. Window Increase Function

   CUBIC maintains the acknowledgment (ACK) clocking of Standard TCP by
   increasing the congestion window only at the reception of an ACK. It
   does not make any change to the fast recovery and retransmit of TCP,
   such as TCP-NewReno [RFC6582] [RFC6675]. During congestion avoidance
   after a congestion event where a packet loss is detected by duplicate
   ACKs or a network congestion is detected by ACKs with ECN-Echo flags
   [RFC3168], CUBIC changes the window increase function of Standard
   TCP. Suppose that W_max is the window size just before the window is
   reduced in the last congestion event.

   CUBIC uses the following window increase function:

$$W\_cubic(t) = C*(t-K)^3 + W\_max \text{ (Eq. 1)}$$

   where C is a constant fixed to determine the aggressiveness of window
   increase in high BDP networks, t is the elapsed time from the
   beginning of the current congestion avoidance, and K is the time
   period that the above function takes to increase the current window
   size to W_max if there are no further congestion events and is
   calculated using the following equation:

$$K = cubic\_root(W\_max*(1-beta\_cubic)/C) \text{ (Eq. 2)}$$

   where beta_cubic is the CUBIC multiplication decrease factor, that
   is, when a congestion event is detected, CUBIC reduces its cwnd to
   W_cubic(0)=W_max*beta_cubic. We discuss how we set beta_cubic in
   Section 4.5 and how we set C in Section 5.

Upon receiving an ACK during congestion avoidance, CUBIC computes the
window increase rate during the next RTT period using Eq. 1. It sets
W_cubic(t+RTT) as the candidate target value of the congestion
window, where RTT is the weighted average RTT calculated by Standard
TCP.

Depending on the value of the current congestion window size cwnd,
CUBIC runs in three different modes.

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

2. The concave region, if CUBIC is not in the TCP-friendly region
    and cwnd is less than W_max.

3. The convex region, if CUBIC is not in the TCP-friendly region and
    cwnd is greater than W_max.

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

4.2. TCP-Friendly Region

Standard TCP performs well in certain types of networks, for example,
under short RTT and small bandwidth (or small BDP) networks. In
these networks, we use the TCP-friendly region to ensure that CUBIC
achieves at least the same throughput as Standard TCP.

The TCP-friendly region is designed according to the analysis
described in [FHP00]. The analysis studies the performance of an
Additive Increase and Multiplicative Decrease (AIMD) algorithm with
an additive factor of alpha_aimd (segments per RTT) and a
multiplicative factor of beta_aimd, denoted by AIMD(alpha_aimd,
beta_aimd). Specifically, the average congestion window size of
AIMD(alpha_aimd, beta_aimd) can be calculated using Eq. 3. The
analysis shows that AIMD(alpha_aimd, beta_aimd) with
alpha_aimd=3*(1-beta_aimd)/(1+beta_aimd) achieves the same average
window size as Standard TCP that uses AIMD(1, 0.5).

    AVG_W_aimd = [alpha_aimd * (1+beta_aimd) /
                 (2*(1-beta_aimd)*p)]^0.5 (Eq. 3)

Based on the above analysis, CUBIC uses Eq. 4 to estimate the window
size W_est of AIMD(alpha_aimd, beta_aimd) with
alpha_aimd=3*(1-beta_cubic)/(1+beta_cubic) and beta_aimd=beta_cubic,
which achieves the same average window size as Standard TCP. When
receiving an ACK in congestion avoidance (cwnd could be greater than

or less than W_max), CUBIC checks whether W_cubic(t) is less than
W_est(t). If so, CUBIC is in the TCP-friendly region and cwnd SHOULD
be set to W_est(t) at each reception of an ACK.

       W_est(t) = W_max*beta_cubic +
                  [3*(1-beta_cubic)/(1+beta_cubic)] * (t/RTT) (Eq. 4)

4.3. Concave Region

When receiving an ACK in congestion avoidance, if CUBIC is not in the
TCP-friendly region and cwnd is less than W_max, then CUBIC is in the
concave region. In this region, cwnd MUST be incremented by
(W_cubic(t+RTT) - cwnd)/cwnd for each received ACK, where
W_cubic(t+RTT) is calculated using Eq. 1.

4.4. Convex Region

When receiving an ACK in congestion avoidance, if CUBIC is not in the
TCP-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 been perturbed 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. In this region, CUBIC is being
very careful by very slowly increasing its window size. The convex
profile ensures that the window increases very slowly at the
beginning and gradually increases its increase rate. 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
(W_cubic(t+RTT) - cwnd)/cwnd for each received ACK, where
W_cubic(t+RTT) is calculated using Eq. 1.

4.5. Multiplicative Decrease

When a packet loss is detected by duplicate ACKs or a network
congestion is detected by ECN-Echo ACKs, CUBIC updates its W_max,
cwnd, and ssthresh as follows. Parameter beta_cubic SHOULD be set to
0.7.

       W_max = cwnd;   // save window size before reduction
       ssthresh = cwnd * beta_cubic; // new slow-start threshold
       ssthresh = max(ssthresh, 2); // threshold is at least 2 MSS
       cwnd = cwnd * beta_cubic;   // window reduction

A side effect of setting beta_cubic to a value bigger than 0.5 is
slower convergence. We believe that while a more adaptive setting of
beta_cubic could result in faster convergence, it will make the
analysis of CUBIC much harder. This adaptive adjustment of
beta_cubic is an item for the next version of CUBIC.

4.6. Fast Convergence

   To improve the convergence speed of CUBIC, we add a heuristic in
   CUBIC. When a new flow joins the network, existing flows in the
   network 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
   bandwidth of the network. To speed up this bandwidth release by
   existing flows, the following mechanism called "fast convergence"
   SHOULD be implemented.

   With fast convergence, when a congestion event occurs, before the
   window reduction of the congestion window, a flow remembers the last
   value of W_max before it updates W_max for the current congestion
   event. Let us call the last value of W_max to be W_last_max.

```
       if (W_max < W_last_max){ // should we make room for others
           W_last_max = W_max;             // remember the last W_max
           W_max = W_max*(1.0+beta_cubic)/2.0; // further reduce W_max
       } else {
           W_last_max = W_max             // remember the last W_max
       }
```

   At a congestion event, if the current value of W_max is less than
   W_last_max, this indicates that the saturation point experienced by
   this flow is getting reduced because of the change in available
   bandwidth. Then we allow this flow to release more bandwidth by
   reducing W_max further. This action effectively lengthens the time
   for this flow to increase its congestion window because the reduced
   W_max forces the flow to have the plateau earlier. This allows more
   time for the new flow to catch up to its congestion window size.

   The 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, the fast convergence SHOULD
   be disabled.

4.7. Timeout

   In case of timeout, CUBIC follows Standard TCP to reduce cwnd
   [RFC5681], but sets ssthresh using beta_cubic (same as in
   Section 4.5) that is different from Standard TCP [RFC5681].

   During the first congestion avoidance after a timeout, CUBIC
   increases its congestion window size using Eq. 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.

4.8. Slow Start

   CUBIC MUST employ a slow-start algorithm, when the cwnd is no more
   than ssthresh. Among the slow-start algorithms, CUBIC MAY choose the
   standard TCP slow start [RFC5681] in general networks, or the limited
   slow start [RFC3742] or hybrid slow start [HR08] for fast and long-
   distance networks.

   In the case when CUBIC runs the hybrid slow start [HR08], 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 Eq. 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.

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 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:

       AVG_W_cubic = [C*(3+beta_cubic)/(4*(1-beta_cubic))]^0.25 *
                     (RTT^0.75) / (p^0.75) (Eq. 5)

   With beta_cubic set to 0.7, the above formula is reduced to:

       AVG_W_cubic = (C*3.7/1.2)^0.25 * (RTT^0.75) / (p^0.75) (Eq. 6)

   We will determine the value of C in the following subsection using
   Eq. 6.

5.1. Fairness to Standard TCP

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

   Standard TCP 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 Standard TCP in the
   above two types of networks. The following two tables show the
   average window sizes of Standard TCP, HSTCP, and CUBIC. The average
   window sizes of Standard TCP and HSTCP are from [RFC3649]. The
   average window size of CUBIC is calculated using Eq. 6 and the CUBIC
   TCP-friendly region for three different values of C.

```
+--------+----------+-----------+------------+-----------+----------+
|  Loss  | Average  |  Average  |   CUBIC    |   CUBIC   |  CUBIC   |
| Rate P |  TCP W   |  HSTCP W  |  (C=0.04)  |  (C=0.4)  |  (C=4)   |
+--------+----------+-----------+------------+-----------+----------+
| 10^-2  |       12 |        12 |         12 |        12 |       12 |
| 10^-3  |       38 |        38 |         38 |        38 |       59 |
| 10^-4  |      120 |       263 |        120 |       187 |      333 |
| 10^-5  |      379 |      1795 |        593 |      1054 |     1874 |
| 10^-6  |     1200 |     12279 |       3332 |      5926 |    10538 |
| 10^-7  |     3795 |     83981 |      18740 |     33325 |    59261 |
| 10^-8  |    12000 |    574356 |     105383 |    187400 |   333250 |
+--------+----------+-----------+------------+-----------+----------+
```

                               Table 1

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

Loss Rate P	Average TCP W	Average HSTCP W	CUBIC (C=0.04)	CUBIC (C=0.4)	CUBIC (C=4)
10^-2	12	12	12	12	12
10^-3	38	38	38	38	38
10^-4	120	263	120	120	120
10^-5	379	1795	379	379	379
10^-6	1200	12279	1200	1200	1874
10^-7	3795	83981	3795	5926	10538
10^-8	12000	574356	18740	33325	59261

Table 2

Table 2 describes the response function of Standard 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 TCP than HSTCP, especially in networks with a short RTT
where TCP performs reasonably well. For example, in a network with
RTT = 0.01 seconds and p=10^-6, TCP has an average window of 1200
packets. If the packet size is 1500 bytes, then 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 Standard TCP, whereas HSTCP is
about ten times more aggressive than Standard 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 Standard 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 long RTT
and high-bandwidth networks. Based on these observations and our
experiments, we find C=0.4 gives a good balance between TCP-
friendliness and aggressiveness of window increase. Therefore, C
SHOULD be set to 0.4. With C set to 0.4, Eq. 6 is reduced to:

   $AVG\_W\_cubic = 1.054 * (RTT^{0.75}) / (p^{0.75})$ (Eq. 7)

Eq. 7 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 Standard
   TCP under long RTT and high-bandwidth networks.

   The following table shows that to achieve the 10 Gbps rate, Standard
   TCP requires a packet loss rate of 2.0e-10, while CUBIC requires a
   packet loss rate of 2.9e-8.

Throughput(Mbps)	Average W	TCP P	HSTCP P	CUBIC P
1	8.3	2.0e-2	2.0e-2	2.0e-2
10	83.3	2.0e-4	3.9e-4	2.9e-4
100	833.3	2.0e-6	2.5e-5	1.4e-5
1000	8333.3	2.0e-8	1.5e-6	6.3e-7
10000	83333.3	2.0e-10	1.0e-7	2.9e-8

                                Table 3

   Table 3 describes the required packet loss rate for Standard 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 [HKLRX06] indicate that CUBIC uses the spare
   bandwidth left unused by existing Standard 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 TCP in fast and
   long-distance networks.

5.4. Investigating a Range of Environments

   CUBIC has been extensively studied by using both NS-2 simulation and
   test-bed experiments covering a wide range of network environments.
   More information can be found in [HKLRX06].

   Same as Standard TCP, 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 Standard TCP in fast and long-distance networks, it can
   fill large drop-tail buffers more quickly than Standard TCP and

increase the risk of a standing queue [KWAF17]. In this case, proper
queue sizing and management [RFC7567] could be used to reduce the
packet queuing delay.

5.5. Protection against Congestion Collapse

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

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 RTTs, their throughput ratio is linearly
proportional to the inverse of their RTT ratios. This is true
independent of the level of statistical multiplexing in the link.

5.7. Performance with Misbehaving Nodes and Outside Attackers

This is not considered in the current CUBIC.

5.8. Behavior for Application-Limited Flows

CUBIC does not raise its congestion window size if the flow is
currently limited by the application instead of the congestion
window. In case of long periods when cwnd has not been updated due
to the application rate limit, such as idle periods, t in Eq. 1 MUST
NOT include these 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 congestion, a routing change, or a mobility
event, CUBIC behaves the same as Standard TCP.

5.10. Incremental Deployment

CUBIC requires only the change of TCP senders, and it does not make
any changes to TCP receivers. That is, a CUBIC sender works
correctly with the Standard TCP receivers. In addition, CUBIC does
not require any changes to the routers and does not require any
assistance from the routers.

6. Security Considerations

   This proposal 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

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

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

   [RFC3649] Floyd, S., "HighSpeed TCP for Large Congestion Windows",
              RFC 3649, DOI 10.17487/RFC3649, December 2003,
              <https://www.rfc-editor.org/info/rfc3649>.

   [RFC3742] Floyd, S., "Limited Slow-Start for TCP with Large
              Congestion Windows", RFC 3742, DOI 10.17487/RFC3742, March
              2004, <https://www.rfc-editor.org/info/rfc3742>.

   [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol",
              RFC 4960, DOI 10.17487/RFC4960, September 2007,
              <https://www.rfc-editor.org/info/rfc4960>.

   [RFC5033] Floyd, S. and M. Allman, "Specifying New Congestion
              Control Algorithms", BCP 133, RFC 5033,
              DOI 10.17487/RFC5033, August 2007,
              <https://www.rfc-editor.org/info/rfc5033>.

   [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP
              Friendly Rate Control (TFRC): Protocol Specification",
              RFC 5348, DOI 10.17487/RFC5348, September 2008,
              <https://www.rfc-editor.org/info/rfc5348>.

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

   [RFC6582] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The
              NewReno Modification to TCP's Fast Recovery Algorithm",
              RFC 6582, DOI 10.17487/RFC6582, April 2012,
              <https://www.rfc-editor.org/info/rfc6582>.

   [RFC6675] Blanton, E., Allman, M., Wang, L., Jarvinen, I., Kojo, M.,
              and Y. Nishida, "A Conservative Loss Recovery Algorithm
              Based on Selective Acknowledgment (SACK) for TCP",
              RFC 6675, DOI 10.17487/RFC6675, August 2012,
              <https://www.rfc-editor.org/info/rfc6675>.

   [RFC7567] Baker, F., Ed. and G. Fairhurst, Ed., "IETF
              Recommendations Regarding Active Queue Management",
              BCP 197, RFC 7567, DOI 10.17487/RFC7567, July 2015,
              <https://www.rfc-editor.org/info/rfc7567>.

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

8.2. Informative References

   [CEHRX07] Cai, H., Eun, D., Ha, S., Rhee, I., and L. Xu, "Stochastic
              Ordering for Internet Congestion Control and its
              Applications", In Proceedings of IEEE INFOCOM,
              DOI 10.1109/INFCOM.2007.111, May 2007.

   [FHP00]   Floyd, S., Handley, M., and J. Padhye, "A Comparison of
              Equation-Based and AIMD Congestion Control", May 2000.

   [GV02]   Gorinsky, S. and H. Vin, "Extended Analysis of Binary
              Adjustment Algorithms", Technical Report TR2002-29,
              Department of Computer Sciences, The University of
              Texas at Austin, August 2002.

   [HKLRX06] Ha, S., Kim, Y., Le, L., Rhee, I., and L. Xu, "A Step
              toward Realistic Performance Evaluation of High-Speed TCP
              Variants", International Workshop on Protocols for Fast
              Long-Distance Networks.

   [HR08]   Ha, S. and I. Rhee, "Hybrid Slow Start for High-Bandwidth
              and Long-Distance Networks", International Workshop on
              Protocols for Fast Long-Distance Networks.

   [HRX08]   Ha, S., Rhee, I., and L. Xu, "CUBIC: A New TCP-Friendly
              High-Speed TCP Variant", ACM SIGOPS Operating System
              Review, DOI 10.1145/1400097.1400105, July 2008.

   [K03]   Kelly, T., "Scalable TCP: Improving Performance in
              HighSpeed Wide Area Networks", ACM SIGCOMM
              Computer Communication Review, DOI 10.1145/956981.956989,
              April 2003.

   [KWAF17]   Khademi, N., Welzl, M., Armitage, G., and G. Fairhurst,
              "TCP Alternative Backoff with ECN (ABE)", Work in
              Progress, draft-ietf-tcpm-alternativebackoff-ecn-05,
              December 2017.

   [XHR04]   Xu, L., Harfoush, K., and I. Rhee, "Binary Increase
              Congestion Control for Fast, Long Distance Networks", In
              Proceedings of IEEE INFOCOM,
              DOI 10.1109/INFCOM.2004.1354672, March 2004.

Acknowledgements

Authors' Addresses

    Injong Rhee
    North Carolina State University
    Department of Computer Science
    Raleigh, NC 27695-7534
    United States of America
    Email: rhee@ncsu.edu

    Lisong Xu
    University of Nebraska-Lincoln
    Department of Computer Science and Engineering
    Lincoln, NE 68588-0115
    United States of America

    Email: xu@unl.edu

    Sangtae Ha
    University of Colorado at Boulder
    Department of Computer Science
    Boulder, CO 80309-0430
    United States of America
    Email: sangtae.ha@colorado.edu

    Alexander Zimmermann
    Phone: +49 175 5766838
    Email: alexander.zimmermann@rwth-aachen.de

    Lars Eggert
    NetApp
    Sonnenallee 1
    Kirchheim 85551
    Germany
    Phone: +49 151 12055791
    Email: lars@netapp.com

    Richard Scheffenegger
    NetApp
    Am Europlatz 2
    Vienna 1120
    Austria
    Email: rs.ietf@gmx.at