mpCUBIC: A CUBIC-like Congestion Control Algorithm for Multipath TCP

Toshihiko Kato(✉), Shiho Haruyama, Ryo Yamamoto, and Satoshi Ohzahata

University of Electro-Communications, Chofu, Tokyo 182-8585, Japan
{kato, hshiho0824, ryo_yamamoto, ohzahata}@net.lab.uec.ac.jp

Abstract. In Multiptah TCP, the congestion control is realized by individual subflows (conventional TCP connections). However, it is required to avoid increasing congestion window too fast resulting from subflows' increasing their own congestion windows independently. So, a coupled increase scheme of congestion windows, called Linked Increase Adaptation, is adopted as a standard congestion control algorithm for subflows comprising a MPTCP connection. But this algorithm supposes that TCP connections use AIMD based congestion control, and if high speed algorithms such as CUBIC TCP are used, the throughput of MPTCP connections might be decreased. This paper proposes a new high speed MPTCP congestion control scheme, mpCUBIC, based on CUBIC TCP.

Keywords: MPTCP · Congestion control · Linked increase adaptation · CUBIC TCP

1 Introduction

Recent mobile terminals are equipped with multiple interfaces. For example, most smart phones have interfaces for 4G Long Term Evolution (LTE) and Wireless LAN (WLAN). However, the conventional Transmission Control Protocol (TCP) establishes a connection between a single IP address at either end, and so it cannot handle multiple interfaces at the same time. In order to utilize the multiple interface configuration, Multipath TCP (MPTCP) [1], which is an extension of TCP, has been introduced in several operating systems, such as Linux, Apple OS/iOS [2] and Android [3]. TCP applications are provided with multiple byte streams through different interfaces by use of MPTCP as if they were working over conventional TCP.

MPTCP is defined in three Request for Comments (RFC) documents standardized by the Internet Engineering Task Force. RFC 6182 [4] outlines architecture guidelines. RFC 6824 [5] presents the details of extensions to support multipath operation, including the maintenance of an MPTCP connection and subflows (TCP connections associated with an MPTCP connection), and the data transfer over an MPTCP connection. RFC 6356 [6] presents a congestion control algorithm that couples the congestion control algorithms running on different subflows.

Á. Rocha et al. (Eds.): WorldCIST 2020, AISC 1160, pp. 306–317, 2020.
https://doi.org/10.1007/978-3-030-45691-7_28

One significant point on the MPTCP congestion control is that, even in MPTCP, individual subflows perform their own control. RFC 6356 requires that an MPTCP data stream do not occupy too large resources compared with other (single) TCP data streams sharing a congested link. For this purpose, it defines an algorithm called Linked Increase Adaptation (LIA), which couples and suppresses the congestion window size of individual subflows. Besides, more aggressive algorithms, such as Opportunistic Linked-Increases Algorithm (OLIA) [7] and Balanced Linked Adaptation (BALIA) [8], are proposed. However, all of those algorithms are based on the Additive Increase and Multiplicative Decrease (AIMD) scheme like TCP Reno [9]. That is, the increase of congestion window at receiving a new ACK segment is in the order of 1/(congestion window size). On the other hand, current operating systems uses high speed congestion control algorithms, such as CUBIC TCP [10] and Compound TCP [11]. These algorithms increase the congestion window more aggressively than TCP Reno. So, it is possible that the throughput of LIA and other MPTCP congestion control algorithms is suppressed when it coexists with them.

We presented the results of performance evaluation in our previous paper [12, 13]. We evaluated the performance when a MPTCP connection with LIA, OLIA or BALIA and a single path TCP with TCP Reno/CUBIC TCP share a bottleneck link. The results showed that the throughput of MPTCP connection is significantly lower than that of CUBIC TCP, and in most cases lower than even TCP Reno.

Based on these results, we propose in this paper a new MPTCP congestion control algorithm which is comparable with CUBIC TCP for single path TCPs. We call this algorithm mpCUBIC (multipath CUBIC), which requires a similar amount of resources as a single path CUBIC TCP, in an MPTCP connection level. This paper proposes the details of mpCUBIC, describes how to implement it over the Linux operating system, and shows the results of performance evaluation. The rest of this paper is organized as follows. Section 2 explains the overview of MPTCP congestion control algorithm. Section 3 describes the procedure of mpCUBIC experimental setting for performance evaluation. Section 4 shows the implementation of mpCUBIC by modifying the CUBIC source program in the Linux operating system. Section 5 shows the results of performance evaluation in an experimental network. In the end, Sect. 6 concludes this paper.

2 Overview of MPTCP Congestion Control

The MPTCP module is located on top of TCP. MPTCP is designed so that the conventional applications do not need to care about the existence of MPTCP. MPTCP establishes an MPTCP connection associated with two or more regular TCP connections called subflows. The management and data transfer over an MPTCP connection is done by newly introduced TCP options for the MPTCP operations.

An MPTCP implementation will take one input data stream from an application, and split it into one or more subflows, with sufficient control information to allow it to be reassembled and delivered to the receiver side application reliably and in order. An MPTCP connection maintains the data sequence number independent of the subflow level sequence numbers. The data sequence number and data ACK number used

in the MPTCP level are contained in a Data Sequence Signal (DSS) option independently from the TCP sequence number and ACK number in a TCP header.

Although the data sequencing, the receipt acknowledgement, and the flow control are performed in the MPTCP level, the congestion control, specifically the congestion window management is performed only in the subflow level. That is, an MPTCP connection does not have its congestion window size. Under this condition, if subflows perform their congestion control independently, the throughput of MPTCP connection will be larger than single TCP connections sharing a bottleneck link. RFC 6356 decides that such a method is unfair for conventional TCP. RFC 6356 introduces the following three requirements for the congestion control for MPTCP connection.

- Goal 1 (Improve throughput): An MPTCP flow should perform at least as well as a single TCP flow would on the best of the paths available to it.
- Goal 2 (Do no harm): All MPTCP subflows on one link should not take more capacity than a single TCP flow would get on this link.
- Goal 3 (Balance congestion): An MPTCP connection should use individual subflow dependent on the congestion on the path.

In order to satisfy these three goals, RC6356 proposes an algorithm that *couples* the additive increase function of the subflows, and uses unmodified decreasing behavior in case of a packet loss. This algorithm is called LIA and summarized in the following way.

Let $cwnd_i$ and $cwnd\_total$ be the congestion window size on subflow i, and the sum of the congestion window sizes of all subflows in an MPTCP connection, respectively. Here, they are maintained in packets. Let rtt_i be the Round-Trip Time (RTT) on subflow i. For each ACK received on subflow i, $cwnd_i$ is increased by

$$\min\left(\frac{\alpha}{cwnd\_total}, \frac{1}{cwnd_i}\right). \tag{1}$$

The first argument of min function is designed to satisfy Goal 2 requirement. Here, α is defined by

$$\alpha = cwnd\_total \cdot \frac{\max_k \left(\frac{cwnd_k}{rtt_k^2}\right)}{\left(\sum_k \frac{cwnd_k}{rtt_k}\right)^2}. \tag{2}$$

By substituting (2) to (1), we obtain the following equation.

$$\min\left(\frac{\max_k \left(\frac{cwnd_k}{rtt_k^2}\right)}{\left(\sum_k \frac{cwnd_k}{rtt_k}\right)^2}, \frac{1}{cwnd_i}\right) \tag{3}$$

As we mentioned in our previous paper, Eq. (2) can be derived by the assumption that the increase and decrease of congestion window sizes are balanced and that the virtual single TCP connections corresponding the subflows have the same throughput as the MPTCP connection [12].

It should be noted that we assume the AIMD scheme in Eqs. (1) and (2). More specifically, we assume that the increase is 1/(congestion window size) for each ACK segment and the decrease parameter is 1/2, which is the specification of TCP Reno. That is, LIA supposes that MPTCP subflows and coexisting single path TCP flows follow TCP Reno. In the case that the high speed congestion control is adopted, the increase per ACK segment will become larger and the decrease parameter will be small.

3 Proposal of mpCUBIC

3.1 Overview of CUBIC TCP

CUBIC TCP determines the value of congestion window size by the following cubic function of the time since the latest congestion detection (last packet loss).

$$W_{cubic}(t) = C(t-K)^3 + W_{max} \tag{4}$$

Here, C is the CUBIC parameter, t is the elapsed time from the last congestion window size reduction, and W_{max} is the congestion window size where the latest loss event occurred. K is the time period that the function (4) takes to increase W_{cubic} to W_{max}, and is given by using the following equation.

$$K = \sqrt[3]{\frac{W_{max} \cdot \beta}{C}} \tag{5}$$

where β is the window decrease constant at a fast retransmit event. As a result, the congestion window changes as shown in Fig. 1 [14]. In this figure, part 1 corresponds to a slow start phase. Part 2 uses only the concave profile of the cubic function from a packet loss until the window size becomes W_{max}. Part 3 uses both the concave and convex profiles, where the convex profile contributes a rapid window increase after the plateau around W_{max}.

Figure 2 shows a balanced situation in CUBIC TCP. The horizontal axis in this figure is time and the vertical axis is a congestion window size. When the window reaches W_{max}, it will reduced due to a packet loss. Then it increases according to the concave profile of the cubic function and it reaches W_{max} again, when the window drops again. The duration between window drops is K. Since the window goes to $W_{max}(1-\beta)$ at its reduction, the following equation is obtained by substituting t = 0 in (4).

$$W_{max}(1-\beta) = C(-K)^3 + W_{max}$$
$$W_{max} \cdot \beta = CK^3$$

This derives (5).

3.2 Design of mpCUBIC

In this paper, we show the design of mpCUBIC in the situation where two subflows are used in an MPTCP connection. When two subflows, both of which use CUBIC TCP, go through one bottleneck link and there are some packet losses due to the congestion, a balanced behavior of congestion window for two subflows is given by Fig. 3. A red line and a black line are a time variation of two CUBIC TCP flows. The feature of the behavior is summarized as follows.

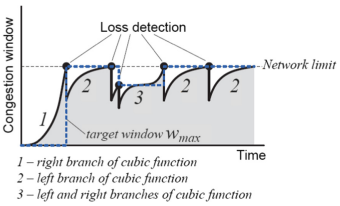


Fig. 1. Cwnd behavior in CUBIC TCP [14].

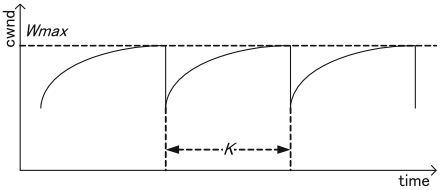


Fig. 2. Balanced cwnd behavior in CUBIC TCP.

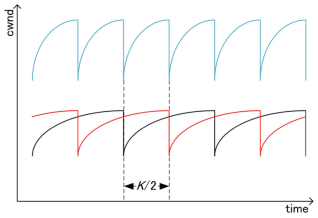


Fig. 3. Balanced cwnd behavior for two CUBIC TCP flows.

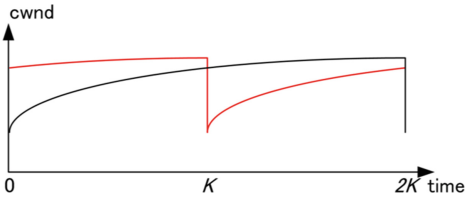


Fig. 4. Congestion window for two mpCUBIC subflows.

- Both of the congestion windows experience packet losses at the same value, W_{max}.
- This behavior is repeated in the same cycle, K
- The cyclic behaviors of two subflows are shifted by $K/2$, that is a half of cycle in the original CUBIC TCP.

As a result, the total congestion window size for an MPTCP connection is given by a blue line in the figure. That is, the total window size also follows a cubic function and the cycle is a half of one subflow.

Therefore, we can deduce the following two points in order to make the total congestion window size for an MPTCP connection with two subflows comparable with single path CUBIC TCP connection.

- The period between two packet losses will be $2K$ instead of K.
- The maximum window size just before a packet loss needs to be set to W_{max} in the total congestion window size for an MPTCP connection.

So, we conclude that the congestion window size of one subflow in mpCUBIC is given by the following equation.

$$W(t) = \frac{1}{D}\left(C'\left(\frac{t}{2} - K\right)^3 + W_{max}\right). \tag{6}$$

Here, C' and D are values defined in mpCUBIC, K and W_{max} are the values used in CUBIC TCP, and t is a time period from the latest packet loss. We focus on one cycle of congestion window size increase for one mpCUBIC subflow together with another subflow, and obtains the graph shown in Fig. 4. The total widow size for one MPTCP connection ($W_{total}(t)$) is given in the following equations.

$$W_{total}(t) = \frac{1}{D}\left(C'\left(\frac{t}{2} - K\right)^3 + C'\left(\frac{t+K}{2} - K\right)^3 + 2W_{max}\right) \text{ for } 0 < t < K \tag{7}$$

$$W_{total}(t) = \frac{1}{D}\left(C'\left(\frac{t}{2} - K\right)^3 + C'\left(\frac{t-K}{2} - K\right)^3 + 2W_{max}\right) \text{ for } K < t < 2K \tag{8}$$

For these two equations, we apply the following requirements.

- At $t = 0$, W_{total} is equal to $(1 - \beta)$ times of the value just before the packet loss. That is,

$$W_{total}(-0)(1 - \beta) = W_{total}(+0). \tag{9}$$

- At $t = 2K$, W_{total} is equal to the maximum value of the conventional CUBIC TCP, Wmax. That is,

$$W_{total}(2K - 0) = W_{max}. \tag{10}$$

From (9), we obtain

$$\left(C'\left(-\frac{1}{2}K\right)^3 + 2W_{max}\right)(1 - \beta) = C'(-K)^3 + C'\left(-\frac{1}{2}K\right)^3 + 2W_{max}.$$

By substituting (5), C' is obtained by the following equation.

$$C' = \frac{2}{1 + \frac{1}{8}\beta}C \tag{11}$$

From (10), we obtain

$$\frac{1}{D}\left(C'\left(-\frac{1}{2}K\right)^3 + 2W_{max}\right) = W_{max}.$$

By substituting (5), D is obtained by the following equation.

$$D = 2 - \frac{\beta}{4\left(1 + \frac{1}{8}\beta\right)} \tag{12}$$

Equations (6), (11) and (12) produce the following equation for the congestion window size of an mpCUBIC sublow.

$$W(t) = \frac{1}{2 - \frac{\beta}{4\left(1+\frac{1}{8}\beta\right)}} \left(\frac{2}{1 + \frac{1}{8}\beta} C \left(\frac{t}{2} - K \right)^3 + W_{max} \right) \tag{13}$$

Here, as described above, C is the CUBIC parameter, t is the elapsed time from the last congestion window size reduction, W_{max} is the congestion window size where the latest loss event occurred, and K is a time period specified by Eq. (5).

4 Implementation

4.1 CUBIC TCP Software in Linux Operating System

In the Linux operating system, the TCP congestion control algorithms are implemented in a form of kernel module. The TCP Reno, NewReno specifically, is implemented in the file `tcp_cong.c`, which is included as the kernel software itself. The other algorithms are implemented in a `tcp_[name of algorithm].c` file, for example, `tcp_cubic.c` for CUBIC TCP. These files are compiled independently of the kernel itself and prepared as kernel modules. The congestion control algorithm that is used in the system is settled by the `sysctl` command setting `net.ipv4.tcp_congestio_control` parameter.

A congestion control program module has the common structure as follows [15].

- Each congestion control program defines the predefined functions, such as how to handle the first ACK segment in a TCP connection (`init()`), how to process a new ACK segment (`pkts_acked()`), and congestion avoidance processing (`cong_avoid()`).
- Each program registers the pointers of the predefined functions described above in the specific data structure defined as `struct tcp_congestion_opt`. In this structure, the name of congestion control algorithm, such as `cubic`, is specified. This name is used in the `sysctl` command.
- Each program calls the `tcp_register_congestion_control` function when it is registered as a kernel module.

In the `tcp_cubic.c` source file, function `bictcp_cong_avoid()` is registered in the `tcp_congestion_opt` data structure as the congestion avoidance function, which calls `bictcp_cong_avoid()` that calculates the congestion window size according Eq. (4).

4.2 Implementation of mpCUBIC

In order to implement mpCUBIC, we modified the `tcp_cubic.c` file as follows.

- In the `tcp_congestion_opt` data structure, we defined a new name, "`mp_cubic`" for our new congestion control algorithm.
- In function `bictcp_cong_avoid()`, the calculation of $(t-K)^3$ is replaced by $(t/2-K)^3$, C is replaced by $2C$, and the result of congestion windows size is multiplied by $1/D$. Here, we used 0.2 as the value of β.

5 Performance Evaluation

5.1 Network Configuration

Figure 5 shows the network configuration used in our experiment. An MPTCP data sender and a TCP (single path TCP) sender are connected to a 100 Mbps Ethernet hub. MPTCP sender has two Ethernet interfaces. A data receiver for MPTCP and TCP is connected to the hub through a bridge, which limits a data transmission rate to 25 Mbps. All nodes are running the Linux operating system, whose distribution is Ubuntu 16.04. Both MPTCP sender and receiver use an MPTCP software whose version is 0.94 stable, which is the newest version we can obtain.

The IP addresses assigned to the network interfaces are shown in Fig. 5. It should be noted that the LAN used in the experiment has subnet address 192.168.0.0/24, and the second Ethernet interface in the MPTCP sender another subnet 192.168.1.0/24. In the MPTCP sender side, the routing table needs to be specified for individual interfaces by using `ip` command. In the receiver side, a route entry to subnet 192.168.1.0/24 needs to be specified explicitly.

In the experiment, one MPTCP connection between the MPTCP sender and the receiver and one TCP connection between the TCP sender and the receiver are evaluated. In order to emulate a wide area Internet communication, 100 ms delay is inserted at the receiver. The bridge limits the data transmission rate from the senders to the receiver into 25 Mbps. Here, an actual congestion will occur. As for the congestion control algorithm, the MPTCP sender uses mpCUBIC, CUBIC, or LIA, and the TCP sender uses CUBIC TCP. The single path TCP uses CUBIC TCP in all cases.

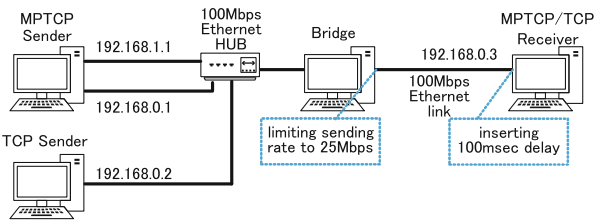


Fig. 5. Network configuration for performance evaluation.

In actual measurement runs, iperf [16] is used for data transfer whose communication duration is 120 s in one measurement run. During the measurement, we obtained communication logs by Wireshark [17], and obtained TCP internal parameter values such as the congestion window size, by tcpprobe [18].

5.2 Evaluation Results

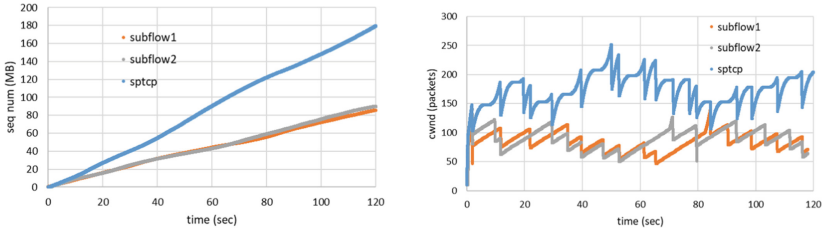
Table 1 gives the results of measured throughput. In the case of mpCUBIC, the MPTCP connection and single path TCP (SPTCP) gave almost the same throughput in a 120 s iperf data transfer. On the other hand, the case of CUBIC, that is, all of two subflows and one SPTCP flow use CUBIC, provided higher throughput for MPTCP than SPTCP. In the case that MPTCP uses LIA, the throughput of MPTCP is largely small compared with CUBIC SPTCP.

Figures 6, 7 and 8 show the time variation of sequence number and congestion window size for the cases that MPTCP uses mpCUBIC, CUBIC, and LIA, respectively. The time variation of sequence number corresponds to the time variation of the number of transferred bytes in two subflows and one SPTCP flow. The congestion window size is measured at each ACK reception by use of tcpporbe.

In mpCUBIC, the time variation of sequence number in two subflows are quite similar and the value is around half of the sequence number in SPTCP. The time variation of congestion window size in two subflows are similar with each other, and is around half of CUBIC SPTCP. These behaviors are exactly what we expected.

Table 1. Throughput results (Mbps).

Scheme	mpCUBIC	CUBIC	LIA
MPTCP	11.7	14.1	3.78
SPTCP	12.0	9.61	20.0

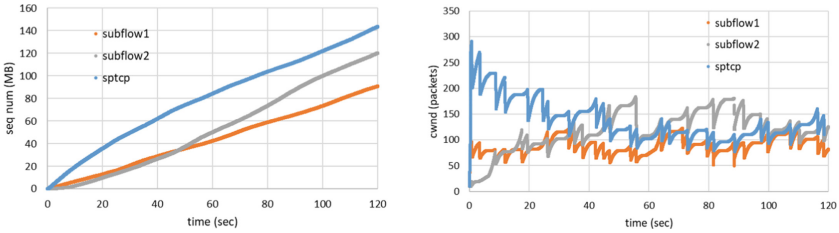


(a) sequence number vs. time (b) cwnd vs. time

Fig. 6. Time variation of sequence number and congestion window size in mpCUBIC.

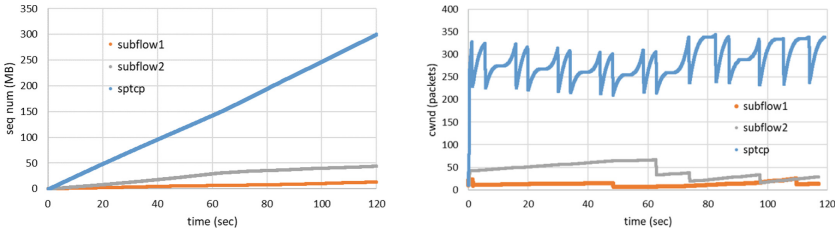
In the case that two MPTCP subflows follow CUBIC TCP, the increases of sequence number in two subflows and one SPTCP are comparable. As a result, the throughput of MPTCP, which is the sum of two subflows' transferred bytes, is larger

than that of one SPTCP. Specifically, the sequence number in SPTCP is larger than those in two subflows. This is because the congestion window size in SPTCP in 30 s at the beginning is larger than MPTCP subflows. The reason is that, in SPTCP, the congestion window size becomes large in the slow start phase just after the connection establishment. During 30 s and 120 s, however, the congestion window sizes in two subflows and one SPTCP are comparable. This is the result that two subflows provide the congestion control independently, and this control provides too favors to MPTCP.



(a) sequence number vs. time (b) cwnd vs. time

Fig. 7. Time variation of sequence number and congestion window size when MPTCP uses CUBIC TCP.



(a) sequence number vs. time (b) cwnd vs. time

Fig. 8. Time variation of sequence number and congestion window size when MPTCP uses LIA.

In the case that MPTCP uses LIA, the increase of sequence number and congestion window size is very small for MPTCP, and CUBIC based SPTCP occupies the bandwidth in the bottleneck link eagerly.

6 Conclusions

This paper proposed a new congestion control algorithm for MPTCP, mpCUBIC, which provides similar performance with CUBIC TCP used by single path TCP connections. The conventional congestion control algorithms for MPTCP, LIA, OLIA, and BALIA, are based on the AIMD scheme, and so they are weaker than high speed

TCP such as CUBIC TCP. Therefore, we modified the CUBIC TCP algorithm so as to fit the multipath environment. For the case that two subflows are used, the cycle of packet losses is doubled and the maximum window for an MPTCP connection is equal to that for a single path TCP.

We implemented mpCUBIC over the Linux operating system, and evaluated the performance over an inhouse testbed network. The performance evaluation results showed that mpCUBIC provides a similar throughput with one CUBIC TCP connection when they share the same bottleneck link. On the other hand, a CUBIC based MPTCP connection requires more resources than a single path CUBIC connection. An LIA MPTCP connection provides poor throughput compared with a single path CUBIC connection.

References

1. Paasch, C., Bonaventure, O.: Multipath TCP. Commun. ACM **57**(4), 51–57 (2014)
2. AppleInsider Staff: Apple found to be using advanced Multipath TCP networking in iOS 7. https://appleinsider.com/articles/13/09/20/apple-found-to-be-using-advanced-multipath-tcp-networking-in-ios-7. Accessed 5 Sep 2019
3. icteam: MultiPath TCP – Linux Kernel implementation, Users:: Android. https://multipath-tcp.org/pmwiki.php/Users/Android. Accessed 5 Sep 2019
4. Ford, A., Raiciu, C., Handley, M., Barre, S., Iyengar, J.: Architectural Guidelines for Multipath TCP Development. IETF RFC 6182 (2011)
5. Ford, A., Raiciu, C., Handley, M., Bonaventure, O.: TCP Extensions for Multipath Operation with Multiple Addresses. IETF RFC 6824 (2013)
6. Raiciu, C., Handley, M., Wischik, D.: Coupled Congestion Control for Multipath Transport Protocols. IETF RFC 6356 (2011)
7. Khalili, R., Gast, N., Popovic, M., Boudec, J.: MPTCP is not pareto-optimal: performance issues and a possible solution. IEEE/ACM Trans. Netw. **21**(5), 1651–1665 (2013)
8. Peng, Q., Valid, A., Hwang, J., Low, S.: Multipath TCP: analysis, design and implementation. IEEE/ACM Trans. Netw. **24**(1), 596–609 (2016)
9. Floyd, S., Henderson, T., Gurtov, A.: The NewReno Modification to TCP's Fast Recovery Algorithm. IETF RFC 3728 (2004)
10. Ha, S., Rhee, I., Xu, L.: CUBIC: a new tcp-friendly high-speed TCP variant. ACM SIGOPS Oper. Syst. Rev. **42**(5), 64–74 (2008)
11. Tan, K., Song, J., Zhang, Q., Sridharan, M.: A compound TCP approach for high-speed and long distance networks. In: IEEE INFOCOM 2006, pp. 1–12. IEEE, Barcelona (2006)
12. Kato, T., Diwakar, A., Yamamoto, R., Ohzahata, S., Suzuki, N.: Performance evaluation of MaltiPath TCP congestion control. In: 18th International Conference on Networks, ICN 2019, pp. 19–24. IARIA, Valencia (2019)
13. Kato, T., Diwakar, A., Yamamoto, R., Ohzahata, S., Suzuki, N.: Experimental analysis of MPTCP congestion control algorithms; LIA, OLIA and BALIA. In: 8th International Conference on Theory and Practice in Modern Computing (TPMC 2019), pp. 135–142. IADIS, Porto (2019)
14. Afanasyev, A., et al.: Host-to-host congestion control for TCP. IEEE Commun. Surv. Tutor. **12**(3), 304–342 (2010)
15. Arianfar, S.: TCP's congestion control implementation in Linux kernel. https://wiki.aalto.fi/download/attachments/69901948/TCP-CongestionControlFinal.pdf. Accessed 5 Nov 2019

16. iperf. http://iperf.sourceforge.net/. Accessed 5 Nov 2019
17. Wireshark. https://www.wireshark.org/. Accessed 5 Nov 2019
18. Linux foundation: tcpprobe. http://www.linuxfoundation.org/collaborate/workgroups/networking/tcpprobe. Accessed 5 Nov 2019