Performance Evaluation of MultiPath TCP Congestion Control

Toshihiko Kato[1)2)], Adhikari Diwakar[1)], Ryo Yamamoto[1)], Satoshi Ohzahata[1)], and Nobuo Suzuki[2)]
1) University of Electro-Communications, Tokyo, Japan
2) Advanced Telecommunication Research Institute International, Kyoto, Japan
e-mail: kato@is.uec.ac.jp, diwakaradh@net.is.uec.ac.jp, ryo_yamamoto@is.uec.ac.jp, ohzahata@is.uec.ac.jp,
nu-suzuki@atr.jp

***Abstract*— In Multiptah TCP (MPTCP), 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 (LIA), is adopted as a standard congestion control algorithm for subflows comprising a MPTCP connection. But this algorithm supposes that TCP connections use Additive Increase and Multiplicative Decrease (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 analyzes this issue through experiments. Specifically, this paper examines two experiments; one is to apply one of LIA, TCP Reno and CUBIC TCP to MPTCP flow, and another is to compare LIA based MPTCP flow and a single TCP flow with TCP Reno or CUBIC TCP. These experiments show that LIA is conservative compared with TCP Reno and CUBIC TCP.**

Keywords- MPTCP; Congestion Control; Linked Increase Adaptation; TCP Reno; CUBIC TCP.

I. INTRODUCTION

Recent mobile terminals are equipped with multiple interfaces. For example, most smart phones have interfaces for 4G Long Term Evolution (LTE) and WLAN. In the next generation (5G) mobile network, it is expected that mobile terminals will be equipped with more interfaces by using multiple communication paths provided multiple network operators [1].

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) [2], which is an extension of TCP, has been introduced in several operating systems, such as Linux, Apple OS/iOS [3] and Android [4]. Conventional TCP applications can use MPTCP as if they were working over conventional TCP and are provided with multiple byte streams through different interfaces.

MPTCP is defined in three Request for Comments (RFC) documents by the Internet Engineering Task Force. RFC 6182 [5] outlines architecture guidelines. RFC 6824 [6] 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 [7] presents a congestion control algorithm that couples the congestion control algorithms running on different subflows.

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 provide too large throughput compared with other (single) TCP data streams sharing a congested link. For this purpose, RFC 6356 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 LIA (OLIA) [8] and Balanced Linked Adaptation (BALIA) [9], are proposed.

However, all of those algorithms are based on the TCP Reno [10]. 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 modern operating systems uses high speed congestion control algorithms, such as CUBIC TCP [11] and Compound TCP [12]. These algorithms increase the congestion window more aggressively than TCP Reno. So, it is possible that the throughput of LIA is suppressed when it coexists with them.

Based on these considerations, we conducted two kinds of experiments. One is for comparing the performance of LIA, the standard congestion control algorithm of MPTCP, with that of the case when subflows use TCP Reno or CUBIC TCP. The other is for evaluating the performance when MPTCP with LIA and TCP Reno / CUBIC TCP share a bottleneck link. This paper describes the results of those experiments.

The rest of this paper is organized as follows. Section II explains the overview of MPTCP and the details of LIA. Here we discuss how LIA algorithm is derived. Section III describes the LIA implementation in the Linux operating system. Section IV shows the performance evaluation of LIA itself and the cases when subflows use TCP Reno or CUBIC TCP. Section V shows the performance evaluation when MPTCP with LIA and TCP with Reno/CUBIC coexist over a bottleneck link. In the end, Section V concludes this paper.

II. OVERVIEW OF MPTCP AND DETAILS OF LIA

A. *Overview of MPTCP*

As described in Figure 1, 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 MPTCP operation.

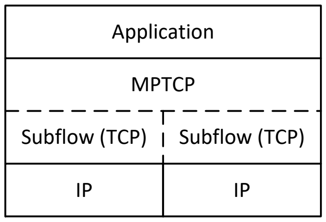


Figure 1. Layer structure of MPTCP.

When the first subflow is established, a TCP option called *MP_CAPABLE* is used within SYN, SYN+ACK, and the following ACK segments. When the following subflows are established, the *MP_JOIN* option is used so that the new TCP connections are associated with the existing MPTCP connection.

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. The MPTCP connection maintains the *data sequence number* independent of the subflow level sequence numbers. The data and ACK segments may contain a *Data Sequence Signal (DSS)* option depicted in Figure 2.

The data sequence number and data ACK is 4 or 8 byte long, depending on the flags in the option. The number is assigned on a byte-by-byte basis similarly with the TCP sequence number. The value of data sequence number is the number assigned to the first byte conveyed in that TCP segment. The data sequence number, subflow sequence number (relative value) and data-level length define the mapping between the MPTCP connection level and the subflow level. The data ACK is analogous to the behavior of the standard TCP cumulative ACK. It specifies the next data sequence number a receiver expects to receive.

B. Overview of MPTCP Congetion Control

As described above, in MPTCP, only subflows manage their congestion windows, 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

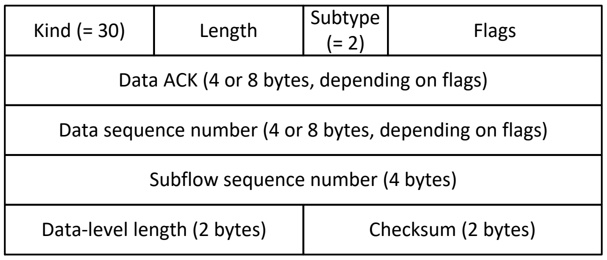

Kind (= 30)
Length
Subtype (= 2)
Flags
Data ACK (4 or 8 bytes, depending on flags)
Data sequence number (4 or 8 bytes, depending on flags)
Subflow sequence number (4 bytes)
Data-level length (2 bytes)
Checksum (2 bytes)

Figure 2. Data Sequence Signal (DSS) option.

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, we assume 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

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

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

$$\alpha = cwnd\_total \cdot \frac{\max_i\left(\frac{cwnd_i}{rtt_i{}^2}\right)}{\left(\sum_i \frac{cwnd_i}{rtt_i}\right)^2}. \quad (2)$$

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

$$min\left(\frac{\max_i\left(\frac{cwnd_i}{rtt_i{}^2}\right)}{\left(\sum_i \frac{cwnd_i}{rtt_i}\right)^2}, \frac{1}{cwnd_i}\right) \quad (3)$$

C. Derivation of LIA Equation

In this subsection, we give one possible derivation of (2), which is not specified in RFC 6356 explicitly. We suppose a single TCP flow corresponding an individual subflow over an MPTCP connection. Let p the packet loss rate over the bottleneck link and let $cwnd_i{}^{TCP}$ be the congestion window size of the supposed single TCP flow i.

We assume the balanced situation indicating that the increase and decrease of congestion window sizes are the same. That is, for subflow i on the MPTCP connection,

$$(1-p)\cdot min\left(\frac{\alpha}{cwnd\_total}, \frac{1}{cwnd_i}\right) = p \cdot \frac{1}{2} cwnd_i. \quad (4)$$

We suppose that the first argument is selected, and then

$$(1-p)\cdot \frac{\alpha}{cwnd\_total} = p \cdot \frac{1}{2} cwnd_i. \quad (4')$$

For supposed TCP flow i,

$$(1-p)\cdot \frac{1}{cwnd_i{}^{TCP}} = p \cdot \frac{1}{2} cwnd_i{}^{TCP}. \quad (5)$$

For satisfying Goals 1 and 2, we can specify

$$\sum_i \frac{cwnd_i}{rtt_i} = \max_i\left(\frac{cwnd_i{}^{TCP}}{rtt_i}\right). \quad (6)$$

By eliminating p using (4') and (5), we obtain

$$\alpha \cdot \left(cwnd_i{}^{TCP}\right)^2 = cwnd\_total \cdot cwnd_i. \quad (7)$$

By squaring both sides of (6) and substituting (7), we obtain

$$\alpha \cdot \left(\sum_i \frac{cwnd_i}{rtt_i}\right)^2 = \max_i\left(\frac{cwnd\_total \cdot cwnd_i}{rtt_i{}^2}\right). \quad (8)$$

This is leading to (2).

It should be noted that we assume the additive increase and multiplicative decrease (AIMD) scheme in (4) and (5). 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 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. In such a case, we need to formalize (4) and (5) in a different way.

III. LIA IMPLEMENTATION OVER LINUX

We can obtain the source program of the Linux operating system including MPTCP from the GitHub web site [13]. We examined how MPTCP are implemented in Linux.

LIA is implemented within the source file `mptcp_coupled.c`. In this file, `mptcp_ccc_recalc_alpha()` and `mptcp_ccc_cong_avoid()` are major functions. The former calculates the first argument in (1) and stores the result in variable `alpha`. The latter records the larger of 1/`alpha` and the congestion window size of the current subflow, and, when this function is called as many times as the recorded value, it increases the congestion window size by one. This procedure is considered to correspond to the specification of (3).

On the other hand, the congestion control mechanisms, strictly speaking the congestion avoidance mechanisms, are implemented as *kernel modules* in Linux. They can be compiled independently of the kernel itself, and can be loaded or removed while the operating system is running. More specifically, the pointer to the function performing congestion avoidance mechanism is stored in a kernel data structure `struct tcp_congestion_ops` within `struct inet_connection_sock` [14]. The kernel function `tcp_cong_control()` calls the function specified in this kernel data structure when it performs congestion avoidance. The pointer to the congestion avoidance function can be settled manually by using `sysctl` command setting control variable `net.ipv4.tcp_congestion_control`. The value will be set to `reno` or `cubic`.

When MPTCP LIA is used, the data structure `struct tcp_congestion_ops` points to the address of function `mptcp_ccc_cong_avoid()` described above. This means LIA is realized as one of TCP congestion avoidance mechanisms. That is, LIA is no automatically selected in MPTCP implementation, but we need to set `net.ipv4.tcp_congestion_control` to `lia` manually. (Or build the kernel to select LIA as a default congestion control algorithm.) In other word, we can use TCP Reno or CUBIC TCP in MPTCP subflows by setting the corresponding control variable.

IV. PERFORMANCE EVALUATION USING PACKET LOSSES

A. Experiment Configuration

As the first experiment, we tried to evaluate the performance of the MPTCP congestion control itself, by generating packet losses artificially. Figure 3 shows the network configuration of the experiment with packet losses inserted. A data sender is connected to 100 Mbps Ethernet and IEEE 802.11g WLAN (2.4 GHz). An 11g access point works as an access point and as an Ethernet hub. A data receiver is connected with the hub through 100 Mbps Ethernet. Both sender and receiver execute MPTCP software with stable version 0.94, which is the newest version [13]. The IP addresses assigned network interfaces of the sender and receiver are shown in Figure 3. The Ethernet interfaces belong to subnet 192.168.0.0/24, and the WLAN interface belongs to another subset 192.168.1.0/24, all of which are connected through a bridge. In the sender side, the routing table need 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. One MPTCP connection with two subflows is established. One subflow goes through the Ethernet interface at the sender, and another goes through the WLAN interface.

The congestion control algorithm used in the sender is set to either of LIA, TCP Reno, or CUBIC TCP. We inserted packet losses with the rate of 0.1% at the Ethernet interface in the sender, and delay of 100 msec at the receiver, both by `tc` (traffic control) command with the `netem` filter. The packets sent through two interfaces at the sender are captured by using *Wireshark* [15], and the congestion window size is recorded for two subflows by using *tcpprobe* [16], both in the sender side. Data transfer is done for 10 sec by *iperf2* [17].

B. Experiment Results

Table I shows the throughput of MPTCP connection measured in two experimental runs for the cases when the congestion control algorithm of MPTCP subflows is set to each of LIA, TCP Reno, and CUBIC TCP. The throughput of LIA, the original setting in MPTCP, is lower than the other settings.

In order to investigate the detail behaviors of individual congestion control algorithms, we examined the time variation of sequence number and congestion window size of MPTCP subflows. Figures 4 through 6 show the results of the experiment runs underlined in Table I. In each algorithm, the congestion window size of a subflow via WLAN interface (WLAN subflow) increases rapidly to its maximum value. It

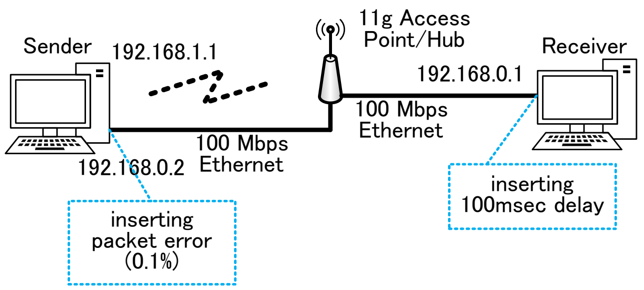
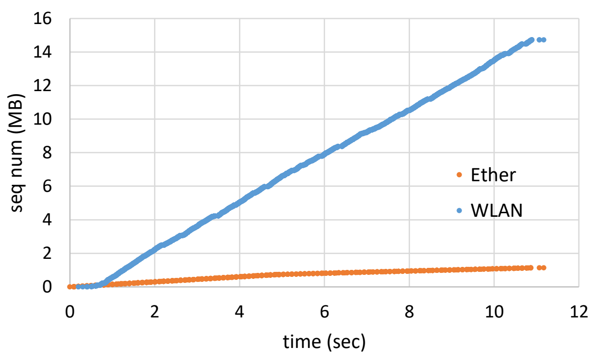


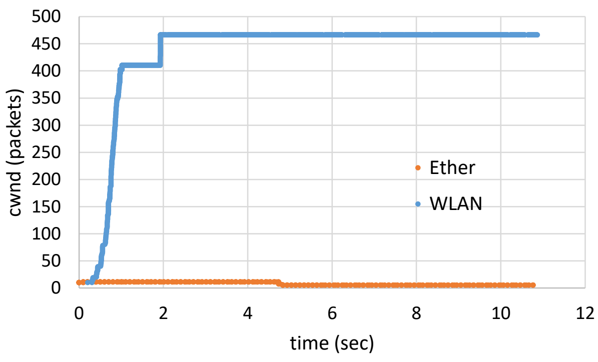
Figure 3. Network configuration by packet loss insertion.

TABLE I. THROUGHPUT WITH PACKET LOSS INSERTED (Mbps).

Algorithm	LIA	Reno	CUBIC
Throughput	<u>12.5</u>, 12.1	14.4, <u>18.8</u>	<u>23.0</u>, 16.1

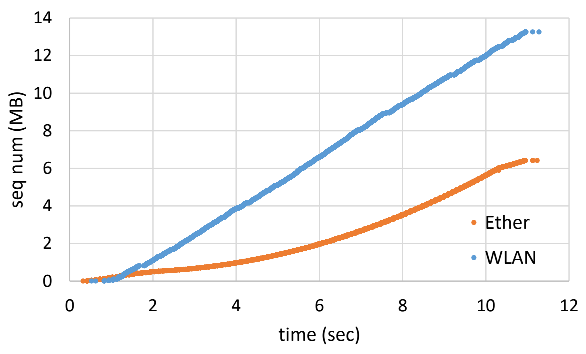


(a) sequence number vs. time

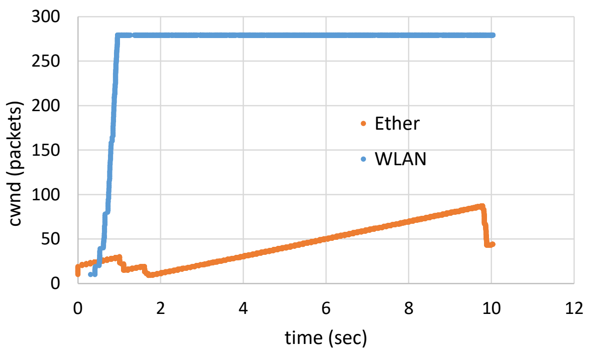


(b) congestion window size vs. time

Figure 4. Time variation of sequence number and congestion window size with packet losses inserted (LIA).

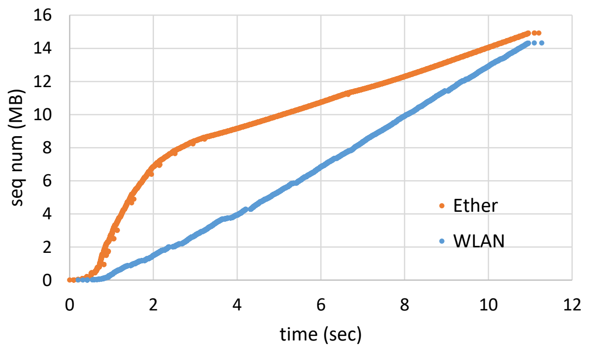


(a) sequence number vs. time

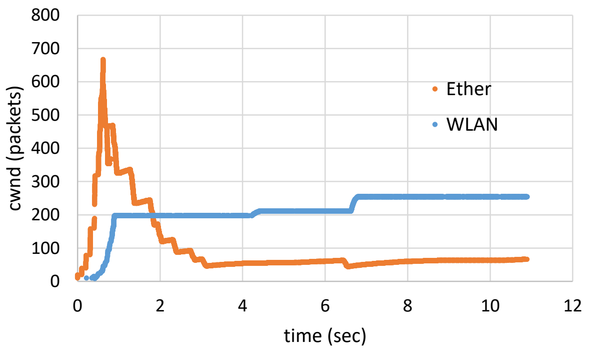


(b) congestion window size vs. time

Figure 5. Time variation of sequence number and congestion window size with packet losses inserted (TCP Reno).



(a) sequence number vs. time



(b) congestion window size vs. time

Figure 6. Time variation of sequence number and congestion window size with packet losses inserted (CUBIC TCP).

should be noted that different maximum values are set to LIA and TCP Reno/CUBIC TCP, by the operating system. It should be also noted that there are some flat parts, before reaching the maximum value, in WLAN congestion window size in the case of CUBIC TCP. The reason for this is supposed that the data corresponding the congestion window size was not sent during one RTT, and that the rule of congestion window validation [18] was applied.

As for a subflow via Ethernet interface (Ethernet subflow), the increase of congestion window size is the smallest in LIA and the largest in CUBIC TCP. So, in the case of LIA, the increase of sequence number, that is, the bytes transmitted, is also limited. In the case that TCP Reno is used as the congestion control algorithm in MPTCP, the congestion window size over Ethernet subflow increases linearly with the elapsed time, which characterizes TCP Reno. The increase is larger than the case of LIA. In the case of CUBIC TCP, the congestion window size over Ethernet subflow increases rapidly by 0.5 sec, and after that it decreases due to several packet losses. During no packet loss period, e.g., from 3 sec to 6.5 sec, we confirmed that the congestion window size changes following a cubic function. Due to the rapid increase during the beginning, the increase of sequence number is large in this case.

For this scenario, it can be said that LIA, the original congestion control algorithm in MPTCP, may be too conservative in increasing congestion window size, compared with TCP Reno and CUBIC TCP, which are commonly used in conventional TCP communications.

V. Performance Evaluation through Actual Congestion

A. Experiment Configuration

As the second experiment, we tried to evaluate the performance of the MPTCP congestion control when there are actual congestion. Figure 7 shows the network configuration used in this experiment. We added a single path TCP data sender and a bridge introducing a bottleneck link in the configuration used in the first experiment. At the interface of the bridge to the data receiver, we set the limit of data link rate to 10 Mbps, by using `tc` command with the `tbf` filter. The reason for limiting the bandwidth to 10 Mbps is that the results in the previous experiment show that the MPTCP throughput is larger than 10 Mbps even if it uses LIA, and so a 10 Mbps link will become a bottleneck actually. The congestion control algorithm at the MPTCP data sender is set to LIA and that at the single path TCP data sender is set to TCP Reno or CUBIC TCP.

B. Experiment Results

Table II shows the average throughput of MPTCP flow and single TCP flow, for 10 sec data transfer by iperf. For each combination of LIA and TCP Reno, or LIA and CUBIC TCP, we conducted four experiment runs. When the single TCP flow uses TCP Reno, the average of four runs is 2.82 Mbps for MPTCP flow and 7.03 Mbps for single TCP flow. When CUBIC TCP is used, that is 1.58 Mbps for MPTCP flow and 8.37 Mbps for single TCP flow. In both cases, the average throughput is lower for MPTCP flow. When the single TCP flow uses CUBIC TCP, the throughput of MPTCP flow is decreased further.

In order to investigate more detailed behaviors, we examined the time variation of sequence number and congestion window size for MPTCP subflows and single TCP flow. We picked up the results indicated by gray shadow in Table II. Figure 8 shows the results when the single TCP subflow uses TCP Reno. The sequence number (transmitted

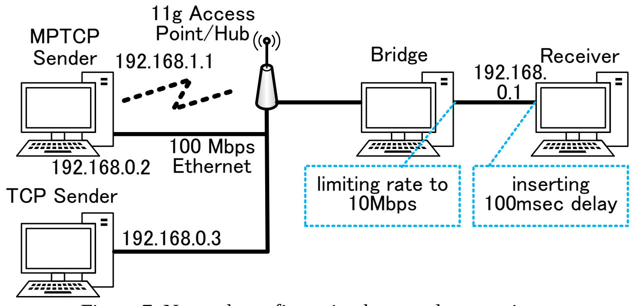
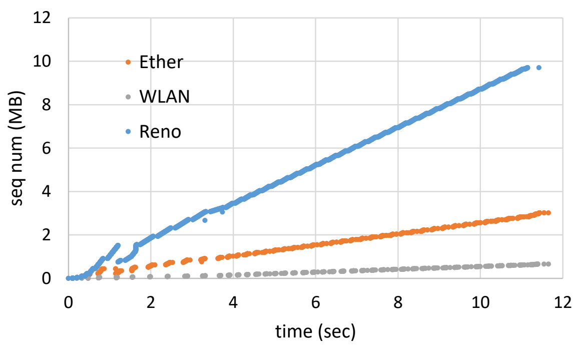


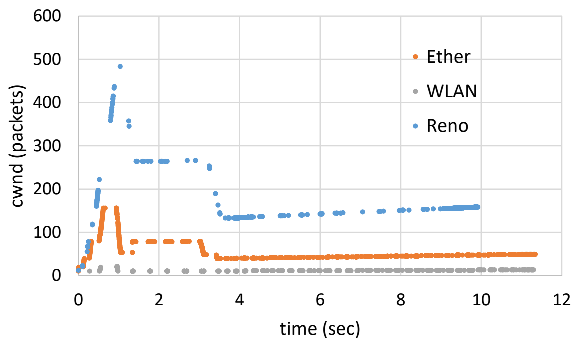
Figure 7. Network configuration by actual congestion.

TABLE II. AVERAGE THROUGHPUT WITH ACTUAL CONGESTION (Mbps).

Algorithm	LIA & Reno				LIA & CUBIC			
MPTCP	2.85	2.66	2.86	2.91	2.03	1.72	1.52	1.04
Single TCP	7.05	7.10	6.97	6.99	7.79	8.33	8.47	8.87

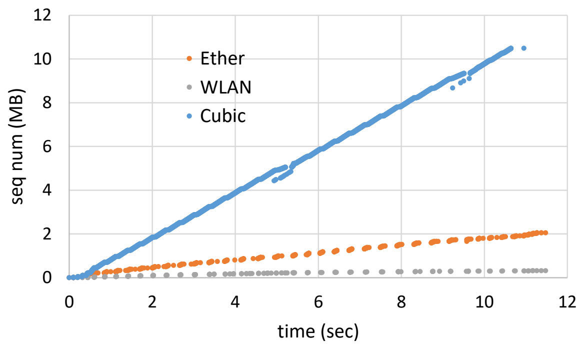


(a) sequence number vs. time

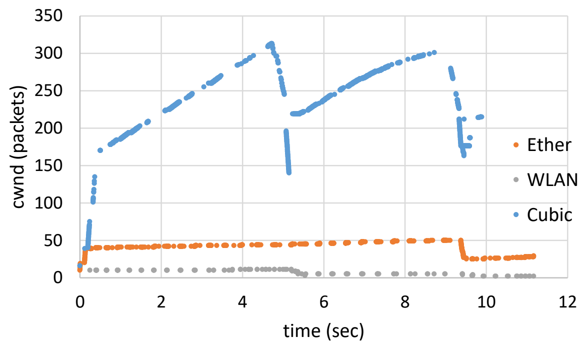


(b) congestion window size vs. time

Figure 8. Time variation of sequence number and congestion window size with actual congestion (LIA & TCP Reno).



(a) sequence number vs. time



(b) congestion window size vs. time

Figure 9. Time variation of sequence number and congestion window size with actual congestion (LIA & CUBIC TCP).

bytes) increases fastest in the single TCP flow, next in the Ethernet subflow and most slowly in the WLAN subflow. As for the time variation of congestion window size, the graph of the single TCP flow and that of the Ethernet subflow are in a similar shape, but the value itself is larger for the single TCP flow. The increase of congestion window size of WLAN subflow is suppressed largely.

Figure 9 shows the results when the single TCP subflow uses CUBIC TCP. In this case, the increase of sequence number is much larger for the single TCP flow. The time variation of congestion window size is also much larger for the single TCP flow. The congestion window size of the MPTCP subflows does not increase but is almost flat along the time. This is similar with the results shown in Figure 4, and this decreases the throughput of MPTCP flow.

From those two results, it can be said that the increase of congestion window in MPTCP subflows using LIA is restricted when they share a congested link with other single TCP flows. The congestion window in LIA is suppressed even when MPTCP subflow shares a bottleneck link with TCP Reno. If LIA coexists with CUBIC TCP, the congestion window is suppressed largely.

VI. Conclusions

This paper described the experimental analysis of the standard congestion control algorithm for MPTCP, Linked Increase Adaptation. As the first experiment, we used a network configuration with Ethernet subflow and WLAN subflow, among which packet losses are inserted in Ethernet subflow. We set the congestion control algorithm for subflows to LIA, TCP Reno, and CUBIC TCP. As a result, the throughput of LIA was smallest. As the second experiment, we used a network configuration using a bridge node introducing a bottleneck link. We also used a node for a single TCP flow. In this configuration, we executed one MPTCP flow using LIA and one single TCP flow with TCP Reno or CUBIC TCP. In this experiment, we obtained a result that MPTCP with LIA is suppressed largely by the single TCP flow with Reno or CUBIC. These results come from the fact that the LIA, the standard congestion control algorithm for MPTCP, is conservative in order to maintain the "Do no harm" principle that requires an MPTCP flow not to use too much network resource compared with single TCP flows. It may be expected to introduce more aggressive congestion control algorithms comparative with high speed congestion control algorithms like CUBIC TPC.

Acknowledgment

This research was performed under the research contract of "Research and Development on control schemes for utilizations of multiple mobile communication networks," for the Ministry of Internal Affairs and Communications, Japan.

References

[1] NGNM Alliance, "NGMN 5G White Paper," https://www.ngmn.org/fileadmin/ngmn/content/downloads/Technical/2015/NGMN_5G_White_Paper_V1_0.pdf, Feb. 2015, [retrieved: Jan. 2019].

[2] C. Paasch and O. Bonaventure, "Multipath TCP," Communications of the ACM, vol. 57, no. 4, pp. 51-57, Apr. 2014.

[3] AppleInsider Staff, "Apple found to be using advanced Multipath TCP networking in iOS 7," http://appleinsider.com/articles/13/09/20/apple-found-to-be-using-advanced-multipath-tcp-networking-in-ios-7, [retrieved: Jan. 2019].

[4] icteam, "MultiPath TCP – Linux Kernel implementation, Users:: Android," https://multipath-tcp.org/pmwiki.php/Users/Android, [retrieved: Jan. 2019].

[5] A. Ford, C.Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural Guidelines for Multipath TCP Development," IETF RFC 6182, Mar. 2011.

[6] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," IETF RFC 6824, Jan. 2013.

[7] C. Raiciu, M. Handley, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," IETF RFC 6356, Oct. 2011.

[8] R. Khalili, N. Gast, M. Popovic, and J. Boudec, "MPTCP Is Not Pareto-Optimal: Performance Issues and a Possible Solution," IEEE/ACM Trans. Networking, vol. 21, no. 5, pp. 1651-1665, Oct. 2013.

[9] Q. Peng, A. Valid, J. Hwang, and S. Low, "Multipath TCP: Analysis, Design and Implementation," IEEE/ACM Trans. Networking, vol. 24, no. 1, pp. 596-609, Feb. 2016.

[10] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno Modification to TCP's Fast Recovery Algorithm," IETF RFC 3728, Apr. 2004.

[11] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," ACM SIGOPS Operating Systems Review, vol. 42, no. 5, pp. 64-74, Jul. 2008.

[12] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A Compound TCP Approach for High-speed and Long Distance Networks," Proc. IEEE INFOCOM 2006, pp. 1-12. Apr. 2006.

[13] GitHub, "Linux Kernel implementation of MultiPath TCP," https://multipath-tcp.org, [retrieved: Jan. 2019].

[14] A. Jaakkola, "Implementation of Transmission Control Protocol in Linux," https://wiki.aalto.fi /download/attachments/70789052/linux-tcp-review.pdf, [retrieved: Jan. 2019].

[15] "Wireshark," https://www.wireshark.org/, [retrieved: Dec. 2018].

[16] Linux Foundation Wiki, "Trace: tcpprobe," The Linux Foundation, https://wiki.linuxfoundation.org/networking/tcpprobe, [retrieved: Jan. 2019].

[17] ESnet, "iperf2/iperf3," https://fasterdata.es.net/performance-testing/network-troubleshooting-tools/iperf/, [retrieved: Jan. 2019].

[18] M. Handley, J. Padhye, and S. Floyd, "TCP Congestion Window Validation," IETF, RFC 2861, Jun. 2000.