Using One-way Communication Delay for In-order Arrival MPTCP Scheduling

Fan Yang
CISC Dept; University of Delaware
Newark, Delaware, USA 19716
yangfan@udel.edu

Paul Amer
CISC Dept; University of Delaware
Newark, Delaware, USA 19716
amer@udel.edu

***Abstract*—We use one-way communication delay of a TCP connection to design an MPTCP scheduler that transmits data out-of-order over multiple paths such that their arrival is in-order. Our Linux implementation shows our proposed scheduler can reduce receive buffer utilization, and increase overall throughput when a small receive buffer size results in receive buffer blocking.**

***Keywords*-MPTCP; multipath TCP; scheduling; transport protocol**

I. Introduction

By simultaneously using multiple TCP paths between peer end hosts [1], Multipath TCP (MPTCP) increases robustness during times of path failure, and potentially achieves higher end-to-end throughput by way of concurrent multipath transfer (CMT) [4]. With its multiple paths, an MPTCP connection has more available network resources than a regular single TCP connection. To be fair, any throughput gain by using MPTCP instead of TCP needs to come from these increased available network resources, and not from consuming more resources in the end hosts. For a fair comparison, a host's send/receive buffer sizes of an MPTCP connection should be same as those of a single TCP connection. Simply, the MPTCP send/receive buffers should be used more efficiently.

Using a send buffer more efficiently means only maintaining 'necessary' data in the send buffer. 'Necessary' data are those packets which have been transmitted but not yet received by the receiver. If a packet is received out-of-order and the receiver guarantees not to discard it, having the sender keep a copy of this packet in its send buffer is 'unnecessary'. Non-Renegable Selective Acknowledgements (NR-SACKs) have been introduced for both SCTP [5] and MPTCP [6]. NR-SACKs allow a receiver to convey non-renegable information of received out-of-order data back to the corresponding sender. The sender in turn can immediately remove NR-SACKed data from the send buffer comfortably knowing the receiver has undertaken the responsibility for that data's eventual delivery.

Using a receive buffer more efficiently can be achieved by decreasing the amount of received out-of-order data. Unlike a TCP connection's single path, an MPTCP connection has multiple possibly asymmetric (i.e., different delays and capacities) paths. Packets can arrive out-of-order not only due to loss or network reordering, but also because of asymmetric paths. Receive buffer blocking has been identified as a problem for multipath data transfer [8, 9]. In the worst case, out-of-order packets may occupy the entire receive buffer causing the entire transmission flow to come to a halt [10].

Three sets of solutions have been proposed to overcome receiver buffer blocking. The first set focuses on the receive buffer, such as using larger receive buffers [8], or dividing the receive buffer into equal blocks [11]. The second is based on modified retransmission policies [9]. Using these policies, a sender tries to provide the receiver with lost in-order packets as fast as possible. The third defines schedulers that transmit packets out-of-order such that they arrive in-order at the receiver [10, 12].

In this work, we define the metric one-way communication delay, and design a scheduler of the third set based on this metric. We implemented an estimation of this metric in the Linux MPTCP kernel [13]. We show that our proposal for in-order arrival scheduling based on one-way communication delay can reduce the receive buffer utilization, and can increase overall throughput when receive buffer blocking occurs for the selected topology. The tradeoff is the added difficulty in practice to measure one-way communication delay.

This paper is organized as follows. Section II defines one-way communication delay and describes a method to measure it. Section III contrasts two ways to implement the scheduler while Section IV provides the pseudo-code of the proposed scheduler implementation. Section V elaborates our test-bed topology, and compares the performance of MPTCP's default scheduler versus our modified scheduler under different scenarios. Section VI provides a brief conclusion.

II. One-Way Communication Delay

Previous delay-aware schedulers have been based on subflows' smoothed RTTs (srtt), essentially using $\mathrm{srtt}/2$ to approximate the one-way communication delay. This assumption has two flaws. First, the forward path delay is not necessarily equal to the return path delay. Second and more importantly, TCP does not include any RTT samples whenever there is a retransmitted TCP-PDU [14]. Thus, the

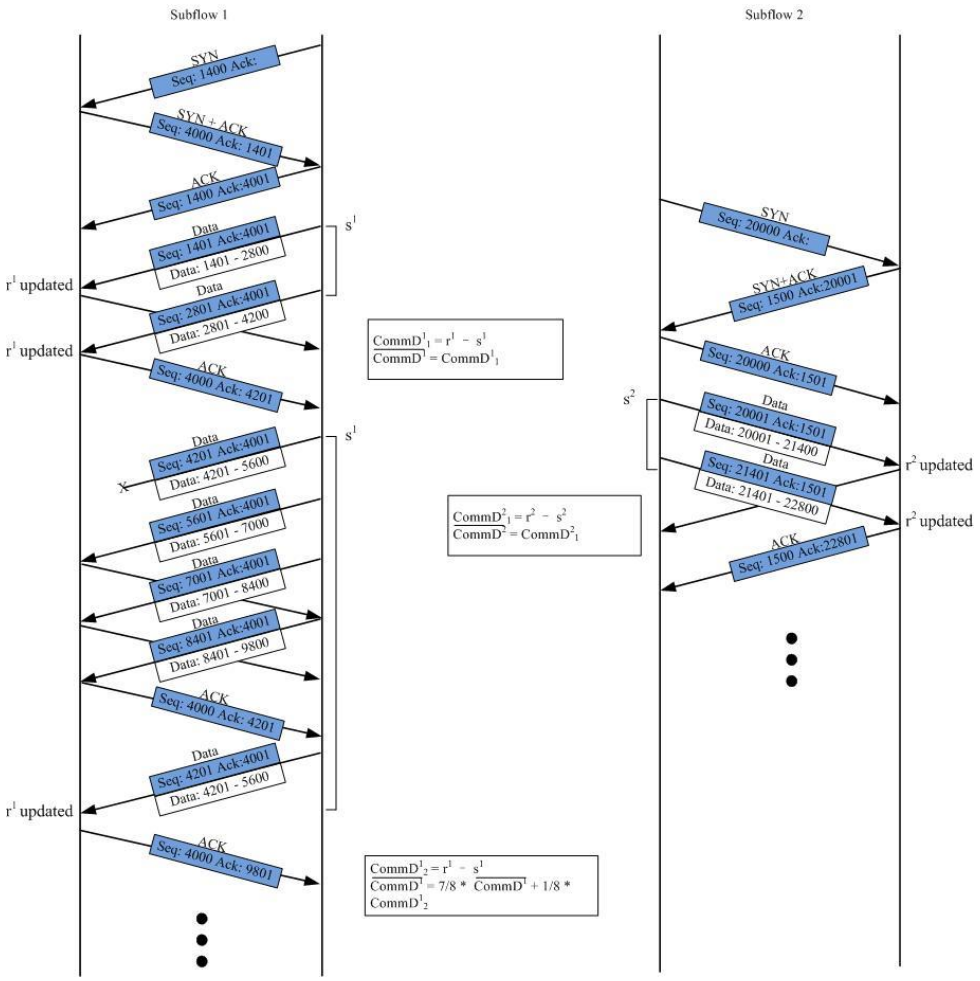

Figure 1. Example of CommD Measurement for MPTCP

srtt measurement in TCP does not reflect increases in one-way communication delay due to losses in the connection. The average one-way communication delay of a connection with a shorter RTT but higher path loss rate can be greater than that of a connection with longer RTT but lower loss rate.

In MPTCP, when a TCP-PDU is received in-order at a subflow's receiver, the payload is delivered immediately to the MPTCP receive buffer. To decrease the amount of received out-of-order data in the MPTCP receive buffer, we propose to focus on the time between when an MPTCP-PDU is sent out on a subflow for the first time, and when that PDU or a retransmission of that PDU first arrives in-order at the subflow's receive buffer (or the MPTCP receive buffer) rather than just the subflows' RTTs. This time interval represents how long it takes for a sender to truly *communicate* its data to the receiver, and is denoted **One-Way Communication Delay** (CommD). CommD is not the same as traditional one-way delay [15, 16, 17] which measures propagation, transmission and queueing delays without taking into account delays due to retransmissions.

A negative aspect of using CommD of a subflow is that it is difficult to measure in practice since the metric is distributed: the start and stop times of the CommD interval occur on different machines.

In our proposed measurement scheme, the end point clocks need not be synchronized. A scheduler only needs to know which subflow has the shortest CommD rather than its actual value, so the scheduler can easily measure a CommD' which is defined as $\mathrm{CommD} + \mathrm{C}$. Here, C is the time difference between the end point clocks.

Let us present a hypothetical example to demonstrate how to measure the CommD' of a subflow in MPTCP. Similar to TCP's measurement of RTT, only one CommD' measurement sample can be in progress at any time. We denote $\mathrm{CommD}'^{\mathrm{i}}_{\mathrm{j}}$ and $\overline{\mathrm{CommD}'^{\mathrm{i}}}$ as the jth measured sample and smoothed average CommD' of subflow i, respectively. Here, 'sample' and 'smoothed' have the analogous meaning as those in RTT measurement. s^{i} is the time when a TCP-PDU (seq: $\mathrm{S_s}$ - $\mathrm{S_e}$) is sent out for the first time, and

is recorded by subflow i's sender as the start time of a sample measurement. Subflow i' receiver constantly updates a variable $\mathrm{r^i}$ to be the time when a PDU is received in-order. The receiver echos the latest value of $\mathrm{r^i}$ to the sender by acknowledgements. The sender pairs the $\mathrm{r^i}$ in the first received acknowledgement (with acknowledgement number $\geq \mathrm{S_e}$) with current $\mathrm{s^i}$. Figure 1 shows an MPTCP connection with two subflows. Two samples are collected for subflow 1, and only one sample is collected for subflow 2.

Note that, the accuracy of $\mathrm{CommD'}$ measurement is influenced by not only delayed acknowledgement but also acknowledgement losses. Compared to RTT, CommD is an one-way delay and accurately accounts for losses of a connection.

III. In-Order Arrival Scheduling

The target of our scheduler is to transmit MPTCP-PDUs on different subflows possibly out-of-order so that they arrive in-order at the MPTCP receive buffer. To achieve this target, two problems need to be solved: one is how to select the next MPTCP-PDU to be scheduled, and the other is on which subflow should this selected MPTCP-PDU be transmitted.

Each MPTCP-PDU needs to be scheduled to a subflow which will make it arrive in-order at the subflow's receiver in the shortest time. We define the time range between when an MPTCP-PDU is scheduled to a subflow and when that PDU arrives in-order at a subflow's receive buffer as the **Delivery Delay** (DeD). DeD differs from CommD. When we talk about CommD, we refer to a TCP connection. While we talk about DeD, we refer to a specific MPTCP-PDU. Note that, for a subflow, 'scheduled' packets may not be sent out immediately because the subflow may not have any available cwnd at scheduling time. Let us use an example to demonstrate the relationship between CommD and DeD.

If MPTCP-PDU i is ready to be scheduled, a scheduler needs to compute $\mathrm{DeD_j^i}$ for each subflow j and schedules MPTCP-PDU i to the subflow with the shortest $\mathrm{DeD_j^i}$. If subflow j has available cwnd, $\mathrm{DeD_j^i = CommD_j}$. Otherwise, $\mathrm{DeD_j^i = n * RTT_j + CommD_j}$. Based on the number of not yet sent packets in the subflow j's send buffer, $\mathrm{n}(\geq 1)$ RTTs are needed before subflow j has available cwnd. We proposed a method to compute n in [12].

A straightforward answer to the first problem is "just select the next not yet scheduled MPTCP-PDU in the MPTCP send buffer". Assume MPTCP-PDU i is selected and scheduled to subflow j. MPTCP-PDU i is encapsulated in a TCP-PDU and placed in subflow j's send buffer. For the default MPTCP scheduler [13], at a given time, only in-flight packets have two copies (one copy is in the MPTCP send buffer and the other is in the subflow send buffer). However, for our proposed scheduler, at a given time, all scheduled packets will have two copies. This first design always schedules packets in-order. Doing so brings a result: a more efficient usage of the receive buffer incurs a less efficient usage of the send buffer. We cannot say this solution is beneficial.

It is preferable to only maintain two copies of in-flight packets, and still achieve in-order arrival. We need to modify the answer to the first problem to be "select an unscheduled MPTCP-PDU which can be sent out now". For example, MPTCP-PDU i is the next as yet unscheduled MPTCP-PDU, and subflow j has the shortest $\mathrm{DeD_j^i}$. If subflow j has available cwnd, MPTCP-PDU i is scheduled and sent out on subflow j. If subflow j has no available cwnd, MPTCP-PDU i is 'assumed' to be scheduled (what we refer to as 'dummy scheduling') to subflow j but will not be copied to its send buffer.

The scheduler continues to consider MPTCP-PDU $\mathrm{i+1}$ until an unscheduled MPTCP-PDU k is found and a subflow l has both available cwnd and the shortest $\mathrm{DeD_l^k}$. Then, MPTCP-PDU k is copied to the send buffer of subflow l and transmitted immediately. Dummy scheduling is necessary to maintain the correctness of the DeD calculation. Compared to the first design, this design schedules packets **out-of-order**. In the next section, we more formally specify the implementation of this design.

IV. Scheduler Implementation

We implement this scheduler in Linux kernel based on the Linux MPTCP [13]. Because of page limitation, the details of the implementation can be found in [7].

V. Performance Evaluation

A. Test-bed Topology

Our test-bed depicted in Figure 2 consists of two Cisco Linksys routers and two laptops running the latest MPTCP kernel. We use Opportunistic Linked Increases Algorithm (OLIA) as the default congestion control mechanism [2]. Both laptops are multihomed by using their tethered Ethernet interface and a Cisco USB Ethernet adapter. An MPTCP connection is established between the two laptops. Subflow 1 is established over the two tethered Ethernet interfaces, while subflow 2 is established between the two Cisco USB Ethernet adapters. Each Cisco USB Ethernet adapter comes with a small internal buffer that can only queue up to 3 packets, thus the intermediate buffer size of subflow 2 is smaller than that of subflow 1. If the intermediate buffers of both subflows are full, the RTT of subflow 2 will be shorter than that of subflow 1. We use FTP to generate MPTCP traffic to confirm the in-order arrival of our proposed scheduler.

B. Receive Buffer Usage

We hypothesized that our proposed scheduler would occupy less receive buffer space than the default scheduler. Figure 3 shows the size of occupied receive buffer size for the default and our proposed schedulers) during the time interval

Table I
THROUGHPUT COMPARISON WITH REDUCED RECEIVE BUFFERS

Receive Buffer (KB)	889	796	707	619	530	442	354	265	177
Default Scheduler (MBps)	1.47	1.47	1.47	1.47	1.47	1.47	1.21	1.20	1.14
In-order Arrival Scheduler (MBps)	1.47	1.47	1.47	1.47	1.47	1.47	1.47	1.47	1.47

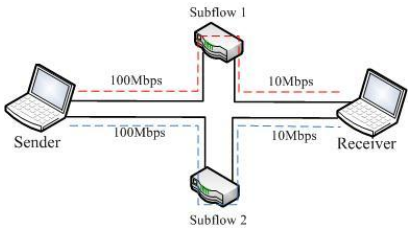

Figure 2. Test-bed Topology

from 20s to 105s of data transfers. The default scheduler can occupy as much as 342KB which is 38% of the entire allocated receive buffer space. The default scheduler reaches the steady state after 95s and occupies 83KB in average. Our proposed scheduler always occupies 25KB, only 2.8% of the allocated buffer space. Our hypothesis is confirmed and our proposal can reduce the usage of the receive buffer for up to 35.2% for this topology.

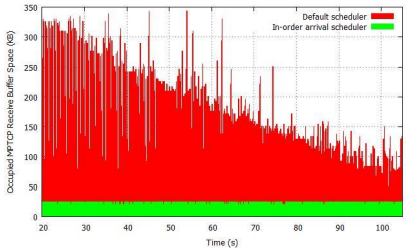

Figure 3. Receive Buffer Usage

C. Throughput with Reduced Receive Buffer

In an MPTCP data transfer, if the sender always has enough traffic to fill all the subflows (i.e., $\mathrm{Sbuf} \geq \sum_{\text{all subflow i}} \bar{w}_i$ (where $\bar{w}_i$ is the average cwnd of subflow i in equilibrium status)) and the receiver always has enough space to accommodate out-of-order packets, the scheduler cannot influence the throughput. However, when the receive buffer decreases, we hypothesize our proposed scheduler will provide greater throughput than the default scheduler. Table I shows the throughput achieved by both schedulers for a variety of receive buffer sizes. We can see the throughput of the default scheduler starts decreasing when the receive buffer is 354KB or smaller, while that with our proposed scheduler remains steady even the receive buffer is reduced to only 177KB.

VI. CONCLUSION AND FUTURE WORK

For our minimal experimentation, we show that, using one-way communication delay, our proposal can reduce the receive buffer utilization for up to 35.2% and increase the throughput when the receive buffer size is smaller than 354KB for the selected topology. By scheduling packets out-of-order, our proposed scheduler does not consume more send buffer space than the default one.

Given these initial positive results, we are planing more extensive testing to further evaluate using our metric of one-way communication delay as a means of performing out-of-order transmission scheduling for in-order arrival. Additionally we are investigating the use of non-renegable selective acknowledgments (NR-SACKs) using statistical methods introduced in [3]. Our target is to make NR-SACKs be added to MPTCP standard.

REFERENCES

[1] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, TCP Extensions for Multipath Operation with Multiple Addresses, RFC 6824, 06/2012

[2] R. Khalili, N. Gast, M. Popovic, U. Upadhyay, J.Y. Le Boudec, *Non Pareto-Optimality of MPTCP: Performance Issues and a Possible Solution*, ACM CoNEXT, 11/2012.

[3] C. Paasch, R. Khalili, O. Bonaventure, *On the Benefits of Applying Experimental Design to Improve MPTCP*, ACM CoNEXT, 12/2013

[4] J. Iyengar, P. Amer, R. Stewart, *Concurrent Multipath Transfer Using SCTP Multihoming over Independent End-to-end Paths*, IEEE/ACM Trans on Networking, 2006

[5] P. Natarajan, N. Ekiz, E. Yilmaz, P. Amer, J. Iyengar, R. Stewart, *Non-Renegable Selective Acks (NR-SACKs) for SCTP*, IEEE International Conference on Network Protocols, Orlando, Florida, 10/2008

[6] F. Yang, P. Amer, *Non-renegable Selective Acks for MPTCP*, PAMS 2013, Barcelona, Spain, 03/2013

[7] F. Yang, *Improving the Performance of MPTCP*, PhD Dissertation, University of Delaware (in progress)

[8] C. Raiciu, C. Paasch, A. Ford, M. Honda, F. Duchene, O. Bonaventure, *How Hard Can it Be? Designing and Implementation a Deployable MPTCP*, USENIX NSDI, 2012

[9] J. Iyengar, P. Amer, R. Stewart, *Retransmission Policies for CMT Using SCTP Multihoming*, IEEE ICON, Singapore, 11/2004

[10] G. Sarwar, R. Boreli, E. Lochin, A. Mifdaoui, G. Smith, *DAPS: Intelligent Delay-Aware Scheduling for Multipath Transport*, ICC, Sydney, Australia, 06/2014

[11] H. Adhari, T. Dreibholz, M. Becke, E. Rathgeb, M. Tüxen, *Evaluation of CMT over Dissimilar Paths*, PAMS 2011, Singapore, 09/2011

[12] F. Yang, Q. Wang, P. Amer, *Out-of-order Transmission for In-order Arrival Scheduling for MPTCP*, PAMS 2014, Victoria, Canada, 05/2014

[13] *MPTCP - Linux Kernel Implementation*, http://mptcp.info.ucl.ac.be

[14] V. Paxson, M. Allman, J. Chu, M. Sargent, Computing TCP's Retransmission Timer, RFC 6298, 06/2011

[15] Q. Pan, X. Luo, H. Xiao, *An Approach to Improve the Accuracy of One-Way Delay Measurements*, Communications in Computer and Information Science, 2011

[16] A. Hernandez, E. Magana, *One-way Delay Measurement and Characterization*, ICNS 2007, Pamplona, Spain, 08/2007

[17] J. Hee, C. Yoo, *One-way delay estimation and its application*, Computer Communications, 2005