Out-of-order Transmission for In-order Arrival Scheduling for Multipath TCP

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

Qi Wang
ECE Dept; U of Delaware
Newark, Delaware, USA 19716
gillwang@udel.edu

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

***Abstract*—MPTCP exploits using a device's multiple interfaces to achieve higher end-to-end throughput and increased robustness during times of path failure. We consider potential delay advantages that MPTCP can provide users of real-time applications (e.g., online gaming) where higher throughput does not necessarily guarantee a higher quality experience. We first explain a situation where jitter can occur in an MPTCP data transfer even when underlying network paths are stable. We propose a new scheduling policy that mitigates jitter by transmitting packets out-of-order on different subflows such that they arrive in-order at the MPTCP receiver.**

***Keywords*-MPTCP; multipath; real-time; scheduling**

I. Introduction

Multipath TCP (MPTCP), perhaps the most significant change to TCP in the past 20 years, simultaneously uses multiple TCP paths between peer end hosts [1]. MPTCP both increases robustness during times of path failure, and potentially achieves higher end-to-end throughput by way of concurrent multipath transfer (CMT) [4].

The full benefit of MPTCP is achieved by a sender which coordinates its scheduler and congestion control mechanisms. A standard MPTCP congestion control mechanism has proposed[2], while the scheduler is still under study.

Currently, a Linux kernel MPTCP implementation [14] is available for real world experiments. Several publications ([9], [10]) point out problems with this implementation, and several schedulers ([5], [6], [7], [13]) have been proposed. Most of these proposed schedulers ([5], [6], [13]) focus on improving the performance in terms of throughput. We consider MPTCP for users of real-time applications (e.g., online gaming), where higher throughput does not necessarily guarantee a higher quality of experience. These real-time applications require stable network service, i.e., low delay variability and jitter [11].

This paper is organized as follows. Section II describes how jitter can occur in MPTCP data transfers even when underlying networks are stable, as well as our thoughts to alleviate this problem. Section III describes a proposed scheduler for out-of-order transmission with in-order arrival. Section IV elaborates our test-bed topology that used to empirically evaluate our proposed scheduler. Section V compares the performance of the default and our modified scheduler under different scenarios. Section VI concludes our work.

II. Problem Description

A. Problem

Consider a hypothetical scenario of an MPTCP connection with two subflows. Both subflows 1 and 2 have cwnd = 4, and the round trip time (RTTs) of subflows 1 and 2 are 200ms and 20ms, respectively. At a given time, 4 MPTCP-PDUs are outstanding on subflow 2, and MPTCP-PDU 5 is ready to be sent. The scheduler must decide on which subflow should MPTCP-PDU 5 be sent?

According to the default scheduler (used by the Linux MPTCP implementation), MPTCP-PDU 5 will be sent on subflow 1 because subflow 2 has no available cwnd. If the application only has 5 MPTCP-PDUs ready to be sent, the total delivery delay of these 5 MPTCP-PDUs would be 100ms, and a time interval of 90ms exists between the delivery time of MPTCP-PDUs 4 and 5. However, if MPTCP-PDU 5 was scheduled to subflow 2, it needs to wait for at most 1 RTT to be sent out. The total delivery delay of these 5 MPTCP-PDUs would be only 30ms and no time interval occurs.

By applying a playout buffer to mitigate the jitter, this situation might not cause a problem for online streaming, where packets are sent out consistently. However, in some applications (e.g., online games), delay is sensitive and packets are generated in short bursts. This time interval can occur periodically and negatively affect the user experience.

A scheduler cannot blindly use both subflows all the time, so it needs to decide when should both subflows be used?

B. Analysis

In a regular TCP connection, if the network does not drop or reorder packets, all TCP-PDUs will arrive in-order at the TCP receiver. While in an MPTCP connection, MPTCP-PDUs will arrive at the MPTCP receiver side out-of-order whenever subflows have different RTTs. The time interval described in the last section is caused by this out-of-order arrival.

Assume the application of the hypothetical scenario in last subsection has 34 MPTCP-PDUs ready to be sent at the beginning. The timeline of data transfer with the default scheduler is shown in Figure 1. The total delivery delay of

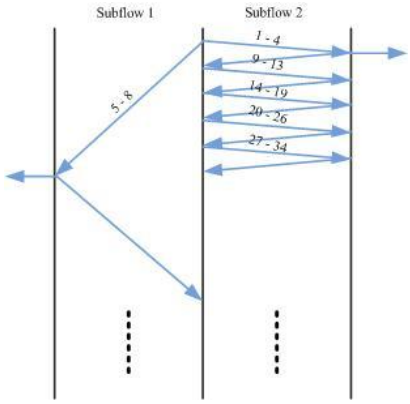

Figure 1. MPTCP Data Transfer with Two Asymmetric Subflows

these 34 MPTCP-PDUs is 100ms, but MPTCP-PDUs 9 to 34 received on subflow 2 cannot be delivered until MPTCP-PDUs 5 to 8 are received on subflow 1.

However, if MPTCP-PDUs 5 to 30 are sent on subflow 2 and MPTCP-PDUs 31 to 34 are sent on subflow 1, the total delivery delay remains 100ms, but all MPTCP-PDUs arrive in-order at the MPTCP receiver and are delivered without a time interval.

How to design such a scheduler? Reconsider the task of a scheduler [13]. Whenever an MPTCP sender wants to send data, the sender needs to make three decisions. First, which subflow(s) can be used (i.e., have available cwnd) to send data? Second, if several subflows have available cwnds, which subflow should be chosen? Third, after selecting a subflow, how much data should be sent on it? The third decision concerns the granularity of the allocation. In this paper, we assume each subflow has the same maximum segment size (MSS), and an MPTCP sender allocates one MSS at a time in step 3. Future work is needed to support other allocation granularities. Here we focus on the scheduler's first two decisions.

Note that all existing schedulers (including the default) only schedule packets to subflows with available cwnds. At any given time, only subflows with available cwnd can be used to schedule data. This default policy is why MPTCP-PDUs 5 to 8 are scheduled to subflow 1 in Figure 1. However, in our proposed scheduler, the answer to the first decision is **all established subflows can be used to schedule data, even those with no available cwnd**.

To make packets arrive in-order at the receiver side, the answer to the second decision is derived as follows. Define T_i^j as the time between when MPTCP-PDU i is scheduled to subflow j and when it arrives at the receiver of subflow j. Note that "MPTCP-PDU i is scheduled to subflow j" does not mean "MPTCP-PDU i can be sent on subflow j immediately", since subflow j may have no available cwnd currently. At any given time, if MPTCP-PDU i is ready to be scheduled and the scheduler knows T_i for all subflows,

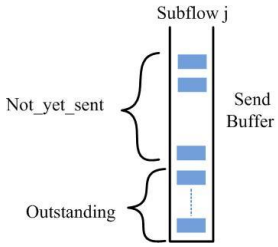

Figure 2. Send Buffer of a Subflow

MPTCP-PDU i will be scheduled to the subflow with the smallest T_i. Each subflow will send out scheduled packets whenever the subflow has available cwnd. With this policy, if the network does not drop or reorder packets, all MPTCP-PDUs will arrive in-order at the MPTCP receiver. Although packets are scheduled in-order, they can be sent out out-of-order. This scheduler provides out-of-order transmission for in-order arrival, an idea suggested for MPTCP in [12].

Our idea is similar to Delay Aware Packet Scheduling (DAPS) ([7], [8]), and we summarize the differences as follow:

- DAPS adapts scheduling when packets are dequeued while our proposed scheduler adapts scheduling when packets are enqueued.
- DAPS uses the information at a given time to determine the scheduling of N packets. Our scheduler uses the information at a given time to determine the scheduling of only 1 packet. Both schedulers use up-to-date information for scheduling.
- DAPS uses a shared send queue architecture. In MPTCP Linux kernel, to preserve the original TCP code, an architecture of two level send queues is used. A packet needs to be moved from a shared send queue to a subflow send queue before that packet can be transmitted.

Currently, DAPS has been implemented in ns-2 for CMT-SCTP, and our scheduler is coded in the Linux MPTCP implementation. Future work is planned to implement DAPS in the Linux kernel to then compare the performance of the two schedulers.

C. Techniques

Now, the problem becomes how to calculate T_i^j for subflow j? The send buffer of a subflow consists of (i) newly scheduled data waiting to be transmitted for the first time, and (ii) data that is outstanding (i.e., already transmitted at least once and awaiting to be cum-acked). In Figure 2, these data are called Not_yet_sent and $Outstanding$, respectively.

For subflow j, $Outstanding_j$ can be $\leq Cwnd_j$ (assume subflow j is not in fast recovery state). Therefore, some

Each time an MPTCP-PDU i is ready to be scheduled:
Min_T = 0xFFFFFFFF

for each subflow k **do**
  if (MPTCP-PDU i is a retransmission) **and** (it has been transmitted on subflow k) **then**
    continue
  end if
  $Num\_available\_path$++
end for

if ($Num\_available\_path > 0$) **then**
  for each available subflow j **do**
    $Number\_of\_packets\_can\_be\_sent_j = Cwnd_j - Outstanding_j$
    $Number\_of\_RTTs\_wait_i^j = \lfloor \frac{Not\_yet\_sent_j - Num\_packets\_can\_be\_sent_j}{Cwnd_j} \rfloor$
    $T_i^j = (Num\_of\_RTTs\_wait_i^j + 0.5) * Srtt_j$
    if ($T_i^j < Min\_T$) **then**
      $Min\_T = T_i^j$
      $Selected\_subflow = j$
    end if
  end for
end if

Figure 3. Algorithm of Out-of-order Transmission for In-order arrival Scheduler

packets can be sent immediately:

$$Number\_of\_packets\_can\_be\_sent_j = Cwnd_j - Outstanding_j \quad (1)$$

The number of RTTs which packet i needs to wait to be sent out:

$$Number\_of\_RTTs\_wait_i^j = \left\lfloor \frac{Not\_yet\_sent_j - Num\_packets\_can\_be\_sent_j}{Cwnd_j} \right\rfloor \quad (2)$$

Therefore,

$$T_i^j = (Num\_of\_RTTs\_wait_i^j + 0.5) * Srtt_j \quad (3)$$

III. Out-of-order transmission for in-order arrival scheduling

When MPTCP-PDU i is ready to be transmitted, the scheduler determines available subflows. If MPTCP-PDU i is an MPTCP level retransmission, it will not be re-sent on the subflow used for the original. Otherwise, all established subflows can be used to send MPTCP-PDU i. If multiple subflows have been established, T_i is calculated for each subflow and the subflow with the smallest T_i is selected. The full scheduling algorithm is given in Figure 3.

To alleviate congestion losses, we use the technique stated in [13] to estimate available path capacities and never over-send packets on a given path.

IV. Evaluation Preliminaries

We implemented our proposed scheduler in the Linux kernel, and evaluated the performance of MPTCP data transfers with two subflows with our proposed scheduler vs. with the default scheduler. The coupled congestion control option is turned on.

Our test-bed consists of two Cisco Linksys routers and two laptops running Ubuntu 11.10 (see Figure 4). Both laptops are multihomed by using the tethered Ethernet interface and a Cisco USB Ethernet adapter. An MPTCP connection is established between the two laptops. Subflow 1 is established over 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. In other words, if the intermediate buffers of both subflows are full, the RTT of subflow 2 will be shorter than that of subflow 1.

Initially we use FTP to generate MPTCP traffic to confirm the in-order arrival of our proposed scheduler. A time-sensitive (e.g., VoIP (Voice over IP), interactive games) application, more appropriate for our evaluation, is under preparation. By comparing user experiences of VoIP with the default and our proposed scheduler, one can evaluate which scheduler is better.

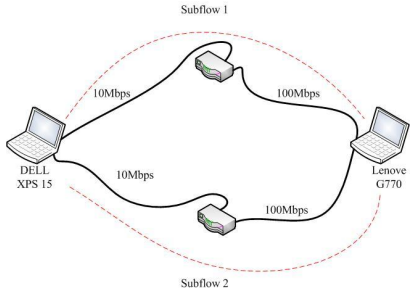

Figure 4. Test-bed Topology

V. Performance Evaluation

A. MPTCP Out-of-order Queue Evolution

Since our scheduler aims to make packets transmitted on different subflows arrive in-order at the MPTCP receiver, we hypothesize that the MPTCP out-of-order queue with our proposed scheduler always contains less data than that with the default during data transfer. Figure 5 shows the sizes of MPTCP level out-of-order queue (with default and our proposed schedulers) between times 20s and 80s of the data transfers. We observe the MPTCP level out-of-order queue size with the default scheduler grows as large as 340KB, while that with our proposed scheduler remains 25KB. Therefore, our first hypothesis is confirmed.

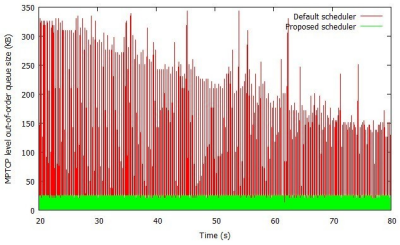

Figure 5. MPTCP Out-of-order Queue during Data transfer

B. Impact of Receive Buffer Size

Since our proposed scheduler can make packets arrive in-order at the MPTCP receiver, our second hypothesis is our proposed scheduler uses the receive buffer more efficiently than the default scheduler. Figure 6 shows the throughputs of our proposed and the default schedulers under different MPTCP receive buffer sizes. We can see the throughput with the default scheduler decreases when the receive buffer is smaller than 460KB, while that with our proposed scheduler remains the same even when the receive buffer is only 160KB. We also note that our proposed scheduler can improve the throughput by never over-sending packets to a subflow.

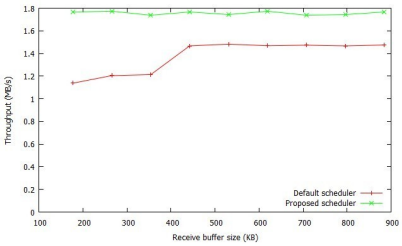

Figure 6. MPTCP Throughputs with Different Receive Buffer Sizes

VI. Conclusion and Future Work

Future work includes comparing our proposed scheduler with a DAPS [8] Linux implementation, and experiments with time-sensitive applications on other topologies and on real network paths. The Simula Research Laboratory (https://simula.no) is currently building a large-scale Internet testbed for multi-homed systems. This NorNet project [15] will be an open platform for researchers doing experimentations with multi-homed systems, i.e. particularly for multipath transport protocols such as MPTCP and CMT-SCTP.

Acknowledgment

The authors would like to thank Jonathan Leighton, Jiefu Li, Emmanuel Lochin and Nichola Kuhn for their valuable suggestions.

References

[1] A. Ford, C. Raiciu, M. Handley, O. Bonaventure, *TCP Extensions for Multipath Operation with Multiple Addresses*, draft-ietf-mptcp-multiaddressed, 6/12

[2] C. Raiciu, M. Handley, D. Wischik, *Coupled Congestion Control for Multipath Transport Protocols*, RFC6356, 10/11

[3] D. Wischik, C. Raiciu, A. Greenhalgh, M. Handley, *Design, Impl and Eval of Congestion Control for MPTCP*. 8th USENIX NSDI, Boston, 3/11

[4] J. Iyengar, P. Amer, R. Stewart, *Concurrent Multipath Transfer Using SCTP Multihoming over Independent End-to-end Paths*. IEEE/ACM Trans on Networking, 14(5), 10/06

[5] A. Singh, C. Goerg, A. Timm-Giel, M. Scharf and T.R. Banniza, *Perf Comparison of Scheduling Algs for Multipath Transfer*. Globecomm, Anaheim, 12/12

[6] W. Zhang, Q. Wu, W. Yang, H. Li, *Reliable Multipath Transfer Scheduling Alg Research and Prototype Impl*. APAN, Sri Lanka, 8/12

[7] G. Sarwar, R. Boreli, E. Lochin, A. Mifdaoui, G. Smith, *Mitigating Receivers Buffer Blocking by DAPS in Multipath Data Transfer*. PAMS, Barcelona, 3/11

[8] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, R. Boreli, *DAPS: Intelligent Delay-Aware Scheduling for Multipath Transport*. ICC, Sydney, 6/14

[9] A.S. Carmelita Görg, A. Timm-Giel, M. Thomas-Ralf Banniza, *Perf Eval of MPTCP Linux Impls*. EuroView, Wurzburg, 8/11

[10] S. Nguyen, X. Zhang, T. Nguyen G. Pujolle, *Eval of Throughput Optimization and Load Sharing of MPTCP in Heterogeneous Nets*. WOCN, New Orleans, 6/11

[11] Y. Chen, Y. Lim, R. Gibbens, E. Nahum, R. Khalili, D. Towsley, *A Measurement-based Study of MPTCP Perf over Wireless Nets*. UMass Tech Rep, 8/13

[12] S. Barré, *Impl and Assessment of Modern Host-based Multipath Solutions*. PhD Dis, U Catholique de Louvain, 2011

[13] F. Yang, P. Amer, N. Ekiz, *A Scheduler for MPTCP*. ICCN, Bahamas, 7/13

[14] *MPTCP-Linux Kernel Impl*. http://mptcp.info.ucl.ac.be

[15] *NorNet Project*. http://www.nornet-testbed.no