Alexander Frömmgen, Jens Heuschkel, Boris Koldehofe
Multipath TCP Scheduling for Thin Streams: Active Probing and One-way Delay-awareness In: Proceedings of the IEEE ICC, May 2018

Multipath TCP Scheduling for Thin Streams: Active Probing and One-way Delay-awareness

Alexander Frömmgen
KOM — TU Darmstadt
alexander.froemmgen@kom.tu-darmstadt.de

Jens Heuschkel
TK — TU Darmstadt
heuschkel@tk.tu-darmstadt.de

Boris Koldehofe
KOM — TU Darmstadt
boris.koldehofe@kom.tu-darmstadt.de

***Abstract*—Multipath TCP (MPTCP) is a recent TCP evolution that uses multiple TCP subflows and thereby different network paths for a single MPTCP connection. The MPTCP *scheduler* has a significant impact on the overall performance of MPTCP, as it maps outgoing packets on TCP subflows.**

In this paper, we identify limitations of today's MPTCP schedulers for thin streams. We present a novel MPTCP scheduler which overcomes today's limitations for thin streams by using *i)* active probing of unused subflows and *ii)* timely one-way delay information. A systematic evaluation within our MPTCP Linux Kernel implementation shows that our novel scheduler outperforms established MPTCP schedulers with regard to application-layer round-trip time without sacrificing efficiency and throughput.

***Index Terms*—Multipath TCP, Scheduling, One-way Delay**

I. Introduction

Multipath TCP is a recent TCP evolution, that allows using multiple TCP subflows for a single logical transport layer connection [7]. This enables mobile devices to concurrently use WiFi and LTE, thereby increasing throughput and resilience [4]. Similarly, MPTCP enables higher throughput and better load balancing in datacenter networks [26].

MPTCP *schedules* packets of one logical transport layer stream on multiple subflows (Figure 1). While details of MPTCP, such as the global sequence numbers and the used options to establish new subflows, are specified [7], scheduling is not standardized. The scheduling decision has a tremendous performance impact [1], as packets on slow subflows may delay flow completion, introduce head-of-line blocking [28], or block further transmission due to receive window restrictions.

Today's *default* MPTCP scheduler assigns packets on the subflow with the lowest round-trip time (RTT) which has not exhausted its congestion window. Long, data-intensive streams, such as heavy elephant streams, are usually throughput limited and easily exhaust the congestion window of the lowest round-trip time subflow. Thus, packets of heavy streams are spread over all subflows by the default scheduler (Figure 1, top). Thin streams, as predominant for interactive applications, such as online games [13], SSH- and control-connections, however, are long running connections which exhibit only a few packets per round-trip time (Figure 1, bottom). These thin streams usually have tight latency requirements and are rather application-layer round-trip time than throughput sensitive. We note that the default scheduler effectively only uses the subflow with the lowest RTT for thin streams, as a few packets easily fit on a single subflow.

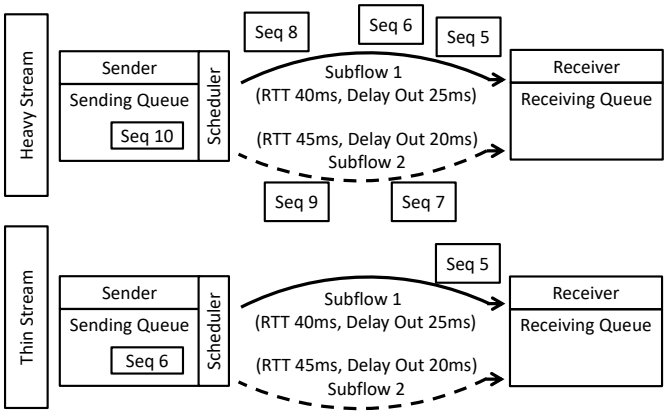

Fig. 1: The MPTCP scheduler assigns packets to subflows. Whereas heavy streams are spread over multiple subflows, thin streams usually rely on the minimum round-trip time subflow unaware of potentially lower one-way delays.[1]

In this paper, we show that today's scheduling for thin streams is limited due to *outdated round-trip time estimates* and missing *one-way delay-awareness*. We present and implement a novel scheduler which overcomes these limitations for thin streams relying only on sender-side modification.

Outdated RTT Estimates: MPTCP implicitly obtains subflow RTT estimates by sending data on these subflows. As thin streams effectively use a single subflow, estimates of the remaining subflows become stale. Thus, the default scheduler is unaware of reducing RTTs of the remaining subflows, e.g., due to fluctuating cross-traffic and queuing [11], [30].

Missing One-Way Delay-Awareness: Round-trip times are seldom symmetric. In datacenters, for example, congestion in a single direction causes asymmetric queuing delays. Furthermore, asymmetric routing in the Internet [24] and properties of access technologies, such as HSDPA, LTE and WiFi [20], cause asymmetric round-trip times. We argue that thin streams, which require low application-layer round-trip times instead of high throughput, benefit from scheduling packets on the subflow with the lowest one-way delay instead of the lowest RTT. One-way delay-aware scheduling even enables application-layer round-trip times below the minimum subflow RTT.

[1] Both sides of the connection run an own instance of the scheduler.

In the remainder of this paper, we present the design of active probing for unused subflows and use established TCP timestamp options to make MPTCP scheduling one-way delay-aware without synchronized clocks. We present MPTCP background in Section II and discuss active probing in Section III. Section IV introduces one-way delay estimations for MPTCP scheduling. Section V presents a detailed evaluation and Section VI concludes the paper.

II. Multipath TCP Background and Related Work

In this section, we present background and related work. Multipath TCP provides the standard TCP socket interface to support unmodified applications. Each MPTCP connection may consist of multiple subflows, where each subflow behaves like a traditional TCP connection. To ensure in-order data delivery on the receiver side, MPTCP transfers control messages such as global sequence numbers in TCP options.

A. Multipath TCP Scheduler

In the Linux Kernel MPTCP implementation [27], the *scheduler* decides per packet on the subflow. The current implementation consists of three schedulers that are denoted *round-robin*, *default* and *redundant*. The *round-robin* scheduler is known to perform poorly for heterogeneous paths, as slow subflows constrain the overall performance [23]. The *default* scheduler assigns packets to the subflow with the lowest RTT that has not exhausted its congestion window [27]. Two *redundant* schedulers were proposed in [9], [21] which transmit each packet on all subflows to reduce latency for applications with moderate bandwidth requirements, and a combination of both is part of to the current MPTCP kernel.

Beside these schedulers in the Linux Kernel MPTCP implementation, additional schedulers where proposed, e.g., for energy-aware or preference-aware video streaming [12], [14] and improved loss compensation for reduced tail flow completion times of short flows in datacenters [3], [16] and to cope with heterogeneous paths [2]. *ProgMP* was recently proposed as programming model for general purpose as well as application- and preference-aware MPTCP scheduling [10], showing a large variety of example schedulers.

The authors of [5], [29] discuss opportunities of multipathing and one-way delay estimations. In contrast, this work is the first which explicitly analyzis MPTCP scheduling limitations for thin flows, i.e., considering the complex dependencies of scheduling, traffic pattern and available scheduling information such as one-way delay estimates, together with an according implementation and detailed evaluation.

B. Round-trip Time Estimations

The default Multipath TCP scheduler relies on the round-trip time estimations of the subflows TCP retransmission timout calculation. Originally, the round-trip time samples for computing TCP's retransmission timer as specified in RFC 6298 [25] relied on Karn's algorithm [18]. However, today the *TCP Timestamp Option* as specified in RFC 1323 (*TCP Extensions for High Performance* [17]) supersede this algorithm and are ubiquitous for round-trip time measurements [19]. Each TCP timestamp option contains two timestamps, the `TSVal` with the senders timestamp and the `TSecr` with the last received timestamps of the communication partner. These values are used to estimate the round-trip time by comparing a replied timestamp (originated from the local clock) with the local time (of the local clock). As only timestamps of the same local clock are compared, clock synchronization is avoided.

III. Active Probing for Thin Streams

Our previous analysis showed that RTT estimates become stale if subflows are not used regularly. Accordingly, we propose to actively probe unused subflows. This probing has to be executed within MPTCP, as alternatives, such as ICMP-based RTT estimates, might be prioritized or routed differently. In the following, we discuss details of our *redundant probing packets at the end of a burst* design.

Redundant Probing Packet: We denote the set of subflows which were not used recently as $\widetilde{\mathtt{S}}$ out of all subflows $\mathtt{S}$. A naive probing approach would schedule *the next* packet of the sending queue on a subflow $s \in \widetilde{\mathtt{S}}$ as soon as there is at least one stale subflow, i.e., $\widetilde{\mathtt{S}} \neq \emptyset$. This probing subflow is in general not the subflow with the minimum round-trip. Thus, scheduling a *fresh* packet on this subflow negatively affects the delay for this packet and might increase the flow completion time. We therefore propose to send the probing packet *redundantly* on the best *and* the stale subflows $\widetilde{\mathtt{S}}$.

At the End of the Burst: Redundancy has to be used carefully to avoid harming performance. Consider a connection is not used for a short time period. Consequently, the scheduler should probe all subflows, as $\widetilde{\mathtt{S}} = \mathtt{S}$. If the application sends a burst of packets, a naive *redundant probing packet* scheduler would transmit the first packet redundantly. If the burst saturates the best subflow, however, the redundant packet wastes resources of the probed subflows. The spare congestion window of the probed subflow should ideally be used for fresh instead of redundant packets. In particular, the last packets of the burst implicitly probe the stale subflows as soon as the best subflow is saturated. We therefore propose to probe stale subflows with redundant packets at the *end of a burst*.

We found ProgMP [10], a programming model for MPTCP scheduling, useful for a concise specification of the active probing scheduler (Figure 2). The highlighted parts (line 17–28) implement active probing in addition to the default scheduler (line 1–15). The scheduler keeps track of the next probing time per subflow (line 20). We use per subflow probing intervals, i.e., a multiple of the subflows RTT, to balance probing aggressiveness for a wide range of environments.

Line 20 checks if the last packet of a burst is scheduled, i.e., if the TCP `PUSH` flag is set. Note that relying on the sending queue to determine the end of a burst (`Q.EMPTY`) is insufficient due to the scheduler timing. When an application pushes data to the socket, the scheduler is invoked as soon as the first packet is queued. Thus, the sending queue `Q` contains a single packet when the scheduler is first called even though the application is still pushing additional data.

```
1  /* Retransmission handling as usual (omitted for simplicity) */
2  ...
3
4  IF (Q.EMPTY) { RETURN; }                          /* No packets in queue => nothing to do */
5
6                                                    /* Retrieve all subflow candidates */
7  VAR candidates = SUBFLOWS.FILTER(sbf => sbf.CWND > sbf.SKBS_IN_FLIGHT + sbf.QUEUED AND
8                      !sbf.THROTTLED AND !sbf.LOSSY AND sbf.HAS_WINDOW_FOR(Q.TOP));
9
10 IF (candidates.EMPTY) { RETURN; }                 /* No subflow candidates => nothing to do */
11
12                                                   /* Schedule as usual. There is at least one subflow */
13 VAR packetToSend = Q.POP();
14 VAR bestSbf = candidates.MIN(sbf => sbf.DELAY_OUT_ESTIMATOR);
15 bestSbf.PUSH(packetToSend);
16
17 VAR probingIntervallRttMultiplier = 5;             /* Tuning parameter */
18
19                                                   /* Reset subflow probing timeout */
20 bestSbf.SET_USER(CURRENT_TIME_MS + bestSbf.RTT * probingIntervallRttMultiplier);
21
22 IF(Q.EMPTY AND packetToSend.PUSH) {               /* End of a burst ? */
23     FOREACH(VAR sbf IN candidates.FILTER(sbf => sbf.USER != 0 AND sbf.USER < CURRENT_TIME_MS)) {
24         /* Send redundant packet and reset subflow probing timeout */
25         sbf.PUSH(packetToSend);
26         sbf.SET_USER(CURRENT_TIME_MS + s.RTT * probingIntervallRttMultiplier);
27     }
28 }
```

Fig. 2: Specification of a one-way delay-aware scheduler with active probing. Line 1–15 essentially represent the default *minRTT* scheduler. In line 14, the scheduler specification relies on the minimum delay estimator calculated according to Section IV. Line 17–28 implement redundant probing packets at the end of a burst.

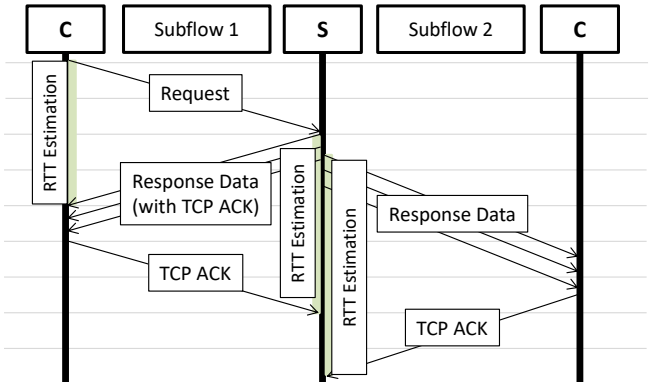

Fig. 3: Even if one direction is not thin, the other direction might not have reasonable RTT estimates.

If the scheduler is allowed to probe, it pushes the last packet of the queue (`packetToSend`) redundantly on stale subflows (line 23–27) and the best currently available subflow (line 15).

Going Beyond Thin Streams: So far, we concentrated on uniform thin streams, which exhibit only a few packet per round-trip time in *both* directions. We note, however, that most heavy streams are actually thin in one direction due to a request-response pattern. Many applications and application layer protocols send small request and large responses in long running TCP connection. For example, HTTP/2 connections handle multiple requests and are kept open for a long time to reduce the handshake overhead [22]. Figure 3 shows an example where a client `C` sends a small request to the server `S`. The server replies with a large response. The data packets and the according acknowledgements enable `S` to estimate the RTT on both subflows. However, `C` is not able to estimate the RTT on the second subflow with the default scheduler, as this requires sending TCP data and receiving a corresponding TCP acknowledgement. Our presented active probing scheduler handles these scenarios efficiently, as it does not introduce overhead for the heavy response and provides RTT estimates on the client side with redundant packets if required.

IV. One-way Delay-Aware MPTCP Scheduling

In this paper, we argue that MPTCP should schedule packets on subflows based on the outgoing one-way delay. One-way delay-aware scheduling has a large impact on thin flows which are sensitive to application-layer round-trip times. However, it has only limited advantage for large flows and does not improve overall throughput, as the number of packet in flight is still limited by the RTT due to the feedback loop with acknowledgements.

One-way delay-aware scheduling essentially requires to order all subflows by their outgoing one-way delay. In the following, we show how to order subflows accordingly relying on the established TCP timestamp options to estimate outgoing one-way delay *differences* of different paths without synchronized clocks. This design enables deployable one-way delay-aware MPTCP scheduling without modifications at the communication partner.

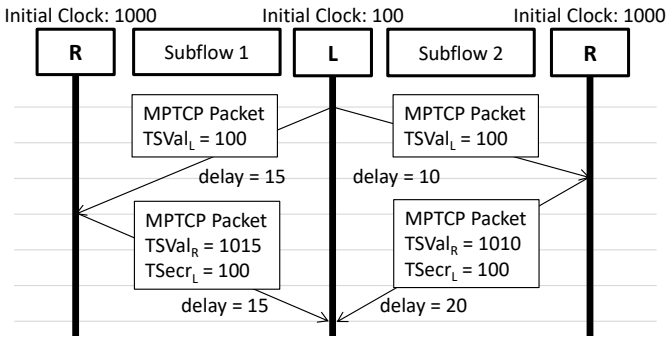

Fig. 4: TCP timestamps for two hosts (L and R) with non-synchronized clocks and two subflows with asymmetric RTTs.

Figure 4 provides a TCP timestamp example with two subflows. Both hosts L and R have different initial clocks (100 vs. 1000). The timestamps of received packets (TSVal and TSecr, as specified in RFC 1323 and introduced in Section II-B) originate from different, non-synchronized clocks. We denote this using the indices L and R for the local and remote clocks. The processing time at the remote host is denoted as P. We assume a reasonable small clock drift between the sender and the receiver clock. We calculate two *delay estimators* DE_{out} and DE_{in} for each subflow

$$\mathrm{DE}_{\text{out}} := \mathrm{TSVal}_R - \mathrm{P} - \mathrm{TSecr}_L \tag{1}$$

$$\mathrm{DE}_{\text{in}} := \mathrm{NOW}_L - \mathrm{TSVal}_R. \tag{2}$$

Due to the non-synchronized clocks, the real delays are not derivable. However, the difference of the outgoing delay estimators of two different subflows corresponds to the time difference when the packets were sent at the remote side. Assuming (on average) constant processing times at the remote host, i.e., $\mathrm{P}_{sbf1} = \mathrm{P}_{sbf2}$, this corresponds to the receive time difference and therefore to the one-way delay difference diff.

$$\begin{aligned}\mathrm{DE}_{\text{out, sbf1}} - \mathrm{DE}_{\text{out, sbf2}} &= \mathrm{TSVal}_{R,\ sbf1} - \mathrm{P}_{sbf1} - \mathrm{TSecr}_{L,\ sbf1} - \\ &\quad (\mathrm{TSVal}_{R,\ sbf2} - \mathrm{P}_{sbf2} - \mathrm{TSecr}_{L,\ sbf2}) \\ &= \mathrm{TSVal}_{R,\ sbf1} - \mathrm{TSecr}_{L,\ sbf1} - \\ &\quad \mathrm{TSVal}_{R,\ sbf2} + \mathrm{TSecr}_{L,\ sbf2}\end{aligned} \tag{3}$$

Note that the initial packets neither have to be sent at the same time nor redundantly. If both packets are sent with a time difference δ, the initial timestamps differ by δ, i.e., $\mathrm{TSecr}_{L,\ sbf1} + \delta = \mathrm{TSecr}_{L,\ sbf2}$. This further implies that the receive time differs by $\delta + \mathrm{diff}$ and $\mathrm{TSVal}_{R,\ sbf1} + \delta + \mathrm{diff} = \mathrm{TSVal}_{R,\ sbf2}$ and therefore $\mathrm{TSVal}_{R,\ sbf1} - \mathrm{TSVal}_{R,\ sbf2} + \delta = \mathrm{diff}$.

In the example (Figure 4), both subflows have the same RTT of 30 but different one-way delays. Host A calculates the *delay estimators* per subflow

$$\mathrm{DE}_{out,\ sbf1} = 1015 - 100 = 915 \tag{4}$$

$$\mathrm{DE}_{out,\ sbf2} = 1010 - 100 = 910. \tag{5}$$

The comparison of both estimators shows that the outgoing delay of the first subflow is higher than the second subflow. Accordingly, the MPTCP scheduler at host A should prefer the second subflow. Remarkably, if both hosts use the one-way delay-aware scheduler, the application layer RTT becomes smaller than the smallest subflow RTT, i.e., $10+15 = 25 < 30$.

We implemented the described delay estimator computation in the `receive_packet_handler` of the existing Linux Kernel MPTCP implementation and integrated it into *ProgMP*s scheduler specification language. This allows the convenient specification of the delay-aware scheduler in Figure 2.

V. Evaluation

In this section, we provide detailed evaluations of our proposed scheduler. The loosely coupled implementation of active probing and one-way delay-aware scheduling in the *ProgMP* MPTCP Linux Kernel enables a detailed evaluation of the features and design decisions.

A. Benchmarking Active Probing

In the following, we evaluate the impact of active probing on the application-layer round-trip time, the efficiency and the maximum achievable throughput.

We systematically reduce the RTT of the second subflow, starting with a high RTT, to replay scenario where congestion initially causes high delays [30] in a Mininet setup with two disjoint paths.[2] We use a sample application which recurrently sends small requests and waits for a response, thereby tracking the application-layer RTT (corresponds to the flow completion time for small flows) as experienced by this application.

Figure 5 shows traces of the application-layer RTT for the established schedulers (default, redundant, and round-robin), our proposed *redundant probing packet* scheduler, and a slight variation which uses fresh probing packets instead redundant packets, as discussed in Section III. The default scheduler constantly uses the first subflow and remains unaware of the second subflows RTT. The redundant scheduler provides the lowest possible application level round-trip time, as the used sample application is not throughput limited. The round-robin scheduler performs bad at the beginning of the trace, as it switches between the low RTT and the high RTT subflow. The detailed analysis shows that the experienced application-layer one-way delay fluctuates (Figure 5, right). Due to a timing dependency of the senders and the receivers round-robin scheduler, the application-layer RTT does not fluctuate.

The *redundant probing packet* scheduler uses the minimum RTT subflow at the beginning. When the second subflows RTT falls below the first ones, the scheduler shows a few smaller application-layer RTT samples caused by the redundant probing packets, before the scheduler eventually prefers the second subflow. The time till the second subflow is preferred is an aggressiveness parameters as result of the round-trip time sample smoothing. Finally, the evaluation shows that the naive non-redundant probing introduces high performance degradations.

[2]We follow established best practises for MPTCP emulations in all Mininet experiments to avoid common emulation pitfalls [8].

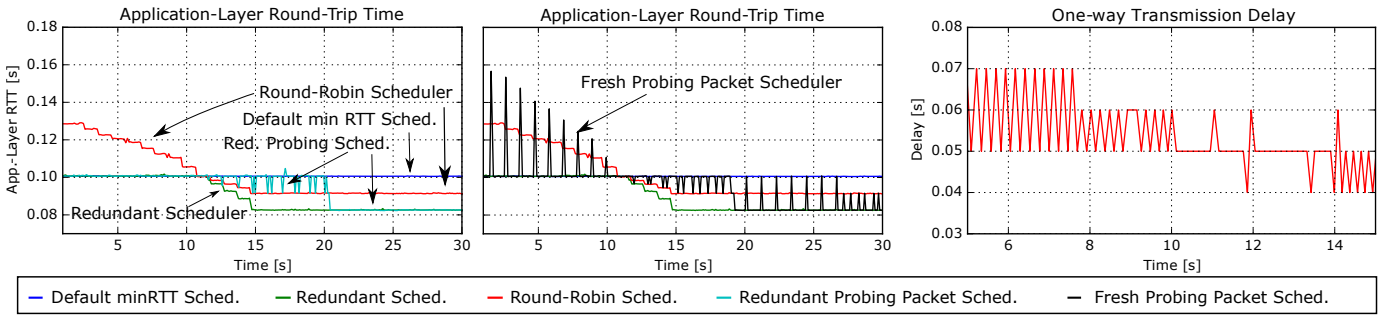

Fig. 5: Application-layer round-trip time comparison. In the Mininet emulation, the first subflow has a constant RTT of 100ms, whereas the RTT of the second subflow decreases from 160ms to 80ms. The measurement shows that the default *minRTT* scheduler is outperformed by redundant and the active probing scheduler.

End of Burst Probing: We further evaluate the design decision to use the *last* packet of a burst for redundant probing. Figure 6 compares the flow completion time for an already established MPTCP connection depending on the flow size in a setup with non-changing 100ms RTT. The evaluation shows that the proposed redundant probing packet at the end of the burst does not harm flow completion time compared to the default *minRTT* scheduler. The naive probing as soon as a subflow becomes stale, however, negatively affects the flow completion time depending on the flow size, i.e., when the redundant packet steals congestion window of fresh packets (for flow sizes between 27.5 and 28.5kb in this example).

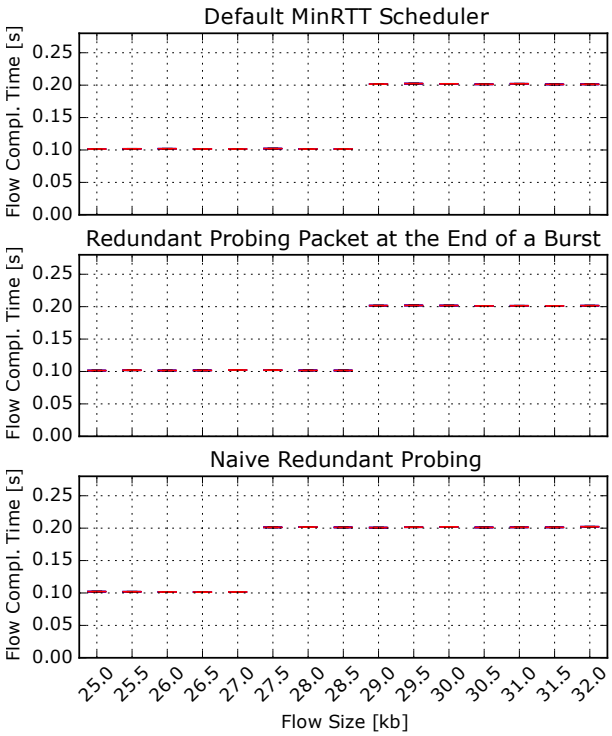

Fig. 6: The proposed redundant probing packet at the end of a burst does not harm flow completion times, whereas a naive probing as soon as a subflow becomes stale negatively affects the flow completion time depending on the flow size.

Overhead: For an evaluation of the induced overhead, we compare the totally transferred data on the wire with the actual data for the previous example of Figure 5. As expected, the redundant scheduler transfers twice the data in the scenario with two subflows (Figure 7). In contrast, the probing scheduler only induces a small overhead. Note that this evaluation shows a single example. In general, the overhead depends on a multitude of aspects, such as the probing interval and the traffic pattern. In particular, the overhead reduces for higher throughput requirements, as shown in the next evaluation.

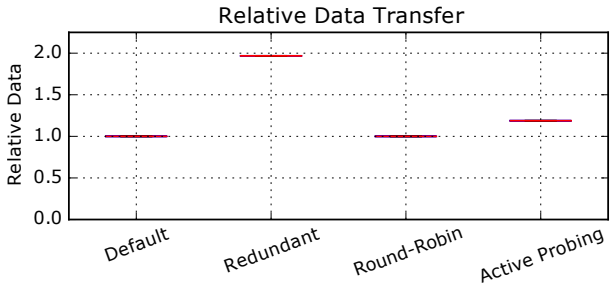

Fig. 7: Normalized data transfer depending on the scheduler.

Throughput: Measurements of the maximum throughput with `iperf` in a scenario with two subflows show only non-significant throughput degradations for the probing scheduler (Figure 8). This is reasonable, as the scheduler does not trigger probing packets for the high throughput workload which saturates all subflows. In contrast, the redundant scheduler

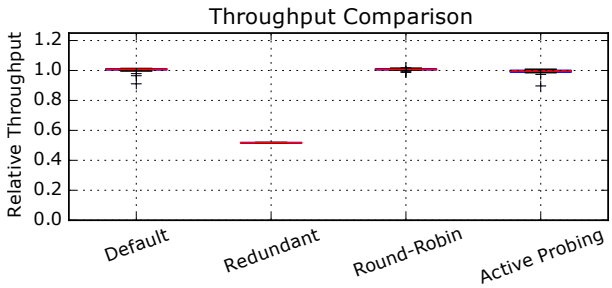

Fig. 8: Throughput comparison with two subflows.

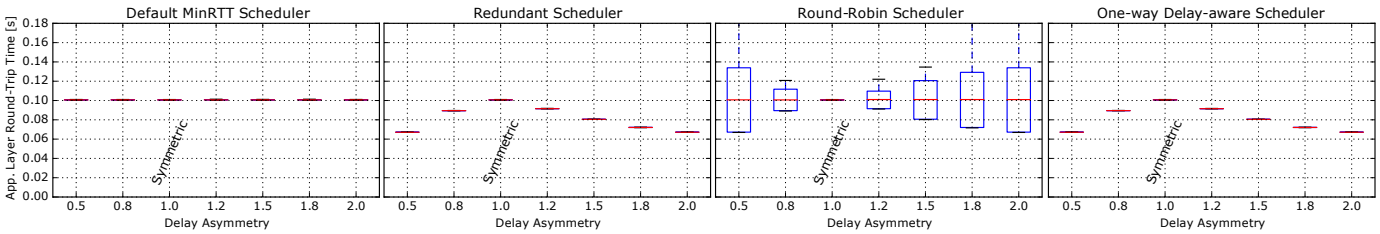

Fig. 9: Application layer round-trip time depending on the delay asymmetry. The default scheduler does not benefit from asymmetric delays, whereas the variance increases for the round-robin scheduler. The one-way delay-aware scheduler leverage the lowest one-way delay subflow. The one-way delay-aware scheduler provides the same reduced application layer round-trip time than the redundant scheduler without inducing overhead.

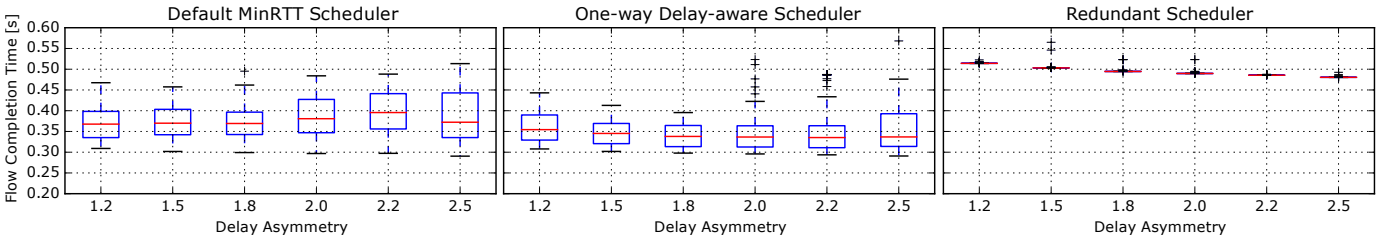

Fig. 10: The default scheduler and the one-way delay-aware scheduler provide comparable flow completion times for 500kb responses, whereas the performance of redundant scheduler decreases.

provides only half the throughput. As a matter of fairness, we note that the redundant scheduler is designed to sacrifice throughput for lowest possible latency of small flows. In particular, the redundant scheduler outperforms all other schedulers with regard to the achieved latency for applications with low throughput requirements in lossy environments due to its proactive loss recovery design and the invested overhead.

B. Benchmarking One-Way Delay-Aware Scheduling

For a evaluation of the one-way delay-aware scheduling, we use Mininet to systematically vary the delay asymmetry $\lambda = \frac{\text{Delay}_{\text{OUT}}}{\text{Delay}_{\text{IN}}}$ while keeping the round-trip time constant. Figure 9 compares the application-layer RTT of the default scheduler, the round-robin scheduler, the redundant scheduler, and the one-way delay-aware scheduler. The application-layer RTT with the default scheduler is independent of the delay asymmetry. The redundant scheduler benefits from asymmetric delays due to the evaluation setup with limited throughput requirements. The round-robin scheduler provides on average the same application-layer RTT as the default scheduler, but shows increasing variances for higher asymmetries. Finally, the one-way delay-aware scheduler is able to leverage TCP timesamps as presented in Section IV and provides the same, minimum application-layer RTT as the redundant scheduler. The measurement further shows that the one-way delay-aware scheduler achieves application-layer round-trip times below the minimum subflow RTT.

Overhead: The overhead of the redundant scheduler becomes apparent when comparing the flow completion time for larger responses of 500kb (Figure 10) which require multiple round-trip times to complete. While the one-way delay-aware scheduler and the default scheduler provide nearly the same flow completion times, the performance of the redundant scheduler decreases.

VI. Conclusion

In this paper, we presented and implemented the first MPTCP scheduler which *i)* schedules packets based on one-way delay estimates derived with traditional TCP timestamp options and *ii)* actively probes unused subflows. Our novel scheduler is easily deployable with sender-side changes only.

The evaluation showed that active probing provides fresh RTT estimates per subflow for improved scheduling decisions and thereby reduces application-layer round-trip times for thin streams in changing environments. We showed that active probing induces only a small overhead thin flows and no overhead for heavy flows. The detailed comparison with alternative probing designs further showed the importance of a careful scheduler design, i.e., a *redundant probing packet at the end of a burst*. We further showed that one-way delay-aware scheduling improves application-layer round-trip times in environments with asymmetric round-trip times without harming flow completion times of larger flows. Thus, our novel scheduler combines the low application-layer round-trip times of the redundant scheduler and the high throughput of the default *minRTT* scheduler.

We provide a ready to use Linux Kernel with our novel scheduler based on *ProgMP* at https://progmp.net/thinStreams. For future work, we envision the application of our finding for emerging multipath protocols such as [6], [15], [31].

Acknowledgment

This work has been funded by the German Research Foundation (DFG) as part of the projects C2 and B2 in the

Collaborative Research Center (SFB) 1053 MAKI.

REFERENCES

[1] B. Arzani, A. Gurney, S. Cheng, R. Guerin, and B. T. Loo. Impact of Path Characteristics and Scheduling Policies on MPTCP Performance. In *Proceedings of the International Conference Advanced Information Networking and Applications Workshops (WAINA)*, pages 743–748. IEEE, 2014.

[2] S. H. Baidya and R. Prakash. Improving the Performance of Multipath TCP over Heterogeneous Paths using Slow Path Adaptation. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 3222–3227, June 2014.

[3] G. Chen, Y. Lu, Y. Meng, B. Li, K. Tan, D. Pei, P. Cheng, L. L. Luo, Y. Xiong, X. Wang, et al. Fast and Cautious: Leveraging Multi-path Diversity for Transport Loss Recovery in Data Centers. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2016.

[4] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley. A Measurement-based Study of Multipath TCP Performance over Wireless Networks. In *Proceedings of the International Measurement Conference (IMC)*, pages 455–468. ACM, 2013.

[5] M. Coudron, S. Secci, and G. Pujolle. Differentiated pacing on multiple paths to improve one-way delay estimations. In *International Symposium on Integrated Network Management (IM)*, pages 672–678. IEEE, 2015.

[6] Q. De Coninck and O. Bonaventure. Multipath QUIC: Design and Evaluation. In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*. ACM, 2017.

[7] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824.

[8] A. Frömmgen. Mininet/Netem Emulation Pitfalls: A Multipath TCP Scheduling Experience. Technical Report https://progmp.net/mininetPitfalls.html, 2017.

[9] A. Frömmgen, T. Erbshäußer, T. Zimmermann, K. Wehrle, and A. Buchmann. ReMP TCP: Low Latency Multipath TCP. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2016.

[10] A. Frömmgen, A. Rizk, T. Erbshäußer, M. Weller, B. Koldehofe, A. Buchmann, and R. Steinmetz. A Programming Model for Application-defined Multipath TCP Scheduling. In *Proceedings of the ACM/IFIP/USENIX Middleware Conference*, pages 134–146, 2017.

[11] J. Gettys and K. Nichols. Bufferbloat: Dark Buffers in the Internet. *ACM Queue*, 2011.

[12] Y. Go, O. C. Kwon, and H. Song. An energy-efficient HTTP adaptive video streaming with networking cost constraint over heterogeneous wireless networks. *IEEE Transactions on Multimedia*, pages 1646–1657, 2015.

[13] C. Griwodz and P. Halvorsen. The Fun of using TCP for an MMORPG. In *Workshop on Network and Operating Systems support for Digital Audio and Video*. ACM, 2006.

[14] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan. MP-DASH: Adaptive video streaming over preference-aware multipath. In *Proceedings of the International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2016.

[15] J. Heuschkel, A. Frömmgen, J. Crowcroft, and M. Mühlhäuser. VirtualStack: Adaptive Multipath Support through Protocol Stack Virtualization. In *Proceedings of the International Network Conference (INC)*, page 73. Lulu.com, 2016.

[16] J. Hwang, A. Walid, and J. Yoo. Fast coupled retransmission for multipath TCP in data center networks. *IEEE Systems Journal*, 2016.

[17] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. RFC 1323, 1992.

[18] P. Karn and C. Partridge. Improving round-trip time estimates in reliable transport protocols. *ACM SIGCOMM Computer Communication Review*, pages 2–7, 1987.

[19] M. Kühlewind, S. Neuner, and B. Trammell. On the state of ECN and TCP options on the Internet. In *Proceedings of the Conference on Passive and Active Measurements (PAM)*, pages 135–144, 2013.

[20] M. Laner, P. Svoboda, P. Romirer-Maierhofer, N. Nikaein, F. Ricciato, and M. Rupp. A comparison between one-way delays in operating HSPA and LTE networks. In *International Symposium on Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt)*, pages 286–292, May 2012.

[21] I. Lopez, M. Aguado, C. Pinedo, and E. Jacob. SCADA Systems in the Railway Domain: Enhancing Reliability through Redundant MultipathTCP. In *International Conference on Intelligent Transportation Systems*, pages 2305–2310. IEEE, Sept 2015.

[22] P. McManus. HTTP/2 is Live in Firefox. https://bitsup.blogspot.de/2015/02/http2-is-live-in-firefox.html, 2015.

[23] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure. Experimental evaluation of Multipath TCP schedulers. In *ACM SIGCOMM Workshop on Capacity Sharing*, pages 27–32. ACM, 2014.

[24] A. Pathak, H. Pucha, Y. Zhang, Y. C. Hu, and Z. M. Mao. A measurement study of internet delay asymmetry. In *Proceedings of the International Conference on Passive and Active Measurements (PAM)*, pages 182–191. Springer-Verlag, 2008.

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

[26] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath tcp. In *ACM SIGCOMM Computer Communication Review*, pages 266–277. ACM, 2011.

[27] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How Hard can it be? Designing and Implementing a Deployable Multipath TCP. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 29–29. USENIX, 2012.

[28] M. Scharf and S. Kiesel. Nxg03-5: Head-of-line blocking in tcp and sctp: Analysis and measurements. In *IEEE Globecom 2006*, pages 1–5, Nov 2006.

[29] F. Song, H. Zhang, S. Zhang, F. Ramos, and J. Crowcroft. An estimator of forward and backward delay for multipath transport. *University of Cambridge, Computer Laboratory, Technical Report*, 2009.

[30] S. D. Strowes. Passively Measuring TCP Round-trip Times. *ACM Queue*, 2013.

[31] T. Viernickel, A. Frömmgen, A. Rizk, B. Koldehofe, and R. Steimetz. Multipath QUIC: A Deployable Multipath Transport Protocol". In *Proceedings of the IEEE International Conference on Communications (ICC)*, 2018.