An Odd Couple: Loss-Based Congestion Control and Minimum RTT Scheduling in MPTCP

Ralf Lübben
Flensburg University of Applied Sciences
Flensburg, Germany
Email: ralf.luebben@hs-flensburg.de

Johannes Morgenroth
Robert Bosch GmbH
Hildesheim, Germany
Email: johannes.morgenroth2@de.bosch.com

***Abstract*—Selecting the best path in multi-path heterogeneous networks is challenging. Multi-path TCP uses by default a scheduler that selects the path with the minimum round trip time (minRTT). A well-known problem is head-of-line blocking at the receiver when packets arrive out of order on different paths. We shed light on another issue that occurs if scheduling have to deal with deep queues in the network. First, we highlight the relevance by a real-world experiment in cellular networks that often deploy deep queues. Second, we elaborate on the issues with minRTT scheduling and deep queues in a simplified network to illustrate the root causes; namely the interaction of the minRTT scheduler and loss-based congestion control that causes extensive bufferbloat at network elements and distorts RTT measurement. This results in extraordinary large buffer sizes for full utilization. Finally, we discuss mitigation techniques and show how alternative congestion control algorithms mitigate the effect.**

***Index Terms*—bufferbloat, multi-path TCP, buffer dimensioning, congestion control**

I. Introduction

Availability of multiple paths between a source and destination open up the possibility of combining these paths for optimal forwarding of packets. Optimization criteria may vary based on the use cases. For example, such criteria are maximizing throughput, increasing reliability, or minimizing delay. Various approaches exists in networking to combine different paths for a better performance. Link aggregation in Ethernet, Equal Cost Multiple Routing (ECMP) in IP networks, or transport layer approaches as Multi-Path TCP (MPTCP), MPQUIC, the multi-path extensions to Quick UDP Internet Connections (QUIC) [1], [2] and the Stream Control Transport Protocol (SCTP) [3] that implements concurrent multipath transfer in CMT-SCTP [4].

Link aggregation and ECMP often distribute flows over multiple paths but avoid splitting up the flow itself. The reason is that if different paths have unequal latency or bandwidth, massive reordering of packets may degrade the flow performance.

To combine multiple paths, approaches on the transport layer are promising to overcome such limitations. Multi-path protocols on the transport layer, such as CMT-SCTP, MPQUIC, and MPTCP, are aware of heterogeneous path characteristics, e.g. by RTT measurements for individual paths, and allow for reordering of packets at the receiver splitted up before on multiple subflows.

In this paper, we focus on MPTCP [5] and its default scheduler for aggregation of all available paths. MPTCP establishes multiple TCP subflows between a sender and a receiver if multiple paths are available. The default scheduler in MPTCP distributes the packets to these different subflows based on the minimum round trip time (minRTT). First, it fills up the subflow with the minimum RTT. If the path is filled up, indicated by a full utilization of the congestion window (CWND) of that subflow, it takes the subflow with the next higher RTT and so on. The CWND of each subflow is typically controlled by a loss based congestion control protocol as NewReno, Cubic or alternatives designed for MPTCP as LIA, OLIA, Balia or xVegas [6], [7], [8], [9].

A well-known issue for MPTCP is head-of-line (HoL) blocking at the receiver [10]. It stems from in-order delivery of packets from the MPTCP socket to the application. If one packet is missing due to delay or packet loss on one of the paths, subsequent packets have to be queued at the receiver in the MPTCP receive queue until the missing packet arrives. This general issue already occurs for all reliable protocols that deliver packets in order but multi-path protocols worsen it, due to different delays on heterogeneous networks path. To mitigate HoL blocking many different schedulers are designed, e.g. [11], [12], [13].

In this paper, we contribute insights on an additional issue that addresses the buffer size requirements for multi-path protocols at the sender when queues in network are deep. This effect is especially prevalent in cellular networks. As presented, e.g. in [14], these networks include elements with deep queues that can hold more than 1000 packets. As we show in the following sections, deep queues impairs the minRTT scheduling. On one hand, MPTCP is a viable solution, to overcome performance limitation and reliability of cellular networks that are prone to regional variations of network quality and coverage. On the other hand, scheduling is a tough challenge with quickly changing network conditions. In the long run, MPTCP is also applicable for session and service continuity as discussed in [15].

In detail, we show that the send buffer size at the sender is an additional limiting factor for the throughput of all subflows in such networks. In Sec. II-C, we demonstrate that buffer sizes for a full utilization have to be much larger than recommended in [16]. Concretely, the send buffer size must be larger than

doi: 10.1109/LCN44214.2019.8990831

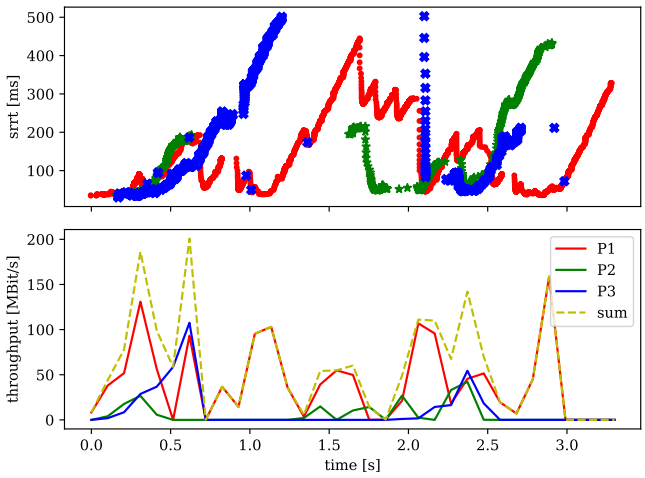

Fig. 1. Throughput and sRTT of MPTCP utilizing three cellular network operators P1, P2, and P3 in a real-world experiment. The sRTT often shows an excessive increase that is about 10 times larger than the minimal RTT of that path. Subsequent to the RTT increase the throughput often drops to zero, which suggests a relation of increase of RTT and throughput degradation.

buffer sizes in the network to overwhelm these buffers and to utilize more than one subflow. The linkage between queue and send buffer sizes arises from loss-based congestion control and is unlocked by the choice of alternative congestion control algorithms.

We highlight the importance by an experiment in real-world cellular networks. The measurement results are shown in Fig. 1. It illustrates the throughput of MPTCP subflows and the sum of all subflows between a greedy sender and a receiver transmitted via three different cellular network operators. The receiver is connected to the server via three different cellular networks. These introductory results in Fig. 1 demonstrate that the throughput of a single subflow drops to zero for prolonged periods which goes along with an excessive increase of RTT. This points out a relation between an increase of delay and limited throughput of the subflows. The understanding of this interaction is crucial since the default scheduler of MPTCP use the RTT as metric to decide on which subflow to send a packet.

To confirm this interaction of minRTT scheduling, loss based congestion control, and deep queues, we contribute in Sec. II-C an in-depth evaluation of the rationales of performance impairment on a basic network scenario derived from the cellular use case. The results shed light on shortcomings regarding RTT measurements, bufferbloat and relating exhaustion of the send buffer size. We further discuss mitigation techniques to improve the performance by congestion control selection and scheduler improvement. The results are transferable to further transport layer multi-path protocols, since protocols implement similar congestion control algorithms, scheduling disciplines, and memory space limitations.

The outline of the paper is as follows: Sec. II gives a detailed evaluation on the performance impairments in networks scenarios that use minRTT scheduling, loss based congestion control in network with deep queues. Sec. III demonstrates how effects are mitigated and performance is improved. Sec. IV presents related work regarding buffer management and congestion control for multi-path transport layer protocols. Sec. V concludes the paper.

II. MinRtt Scheduling and Loss Based Congestion Control

In this section, we present the impact of buffer sizes at the sender and receiver and explain why the buffer space limitations induce the effects explained in Sec. I. First, we give a detailed description of the relation between memory and send buffer size required for the upcoming sections. Second, we describe the experimental setup. Thereafter, we show experimental results of average MPTCP throughput in a simplified scenario that is based on a cellular network. These results highlight the impact of send and receiver buffer dimensioning. We delve into these results and show root causes of throughput degradation by an in-depth analysis of internal parameters read from the MPTCP stack.

A. Send Buffer and Memory Structure

Various windows and buffer spaces regulate the throughput of a TCP flow. The CWND refers to the number of packets to be sent out without overloading the path, the receive window advertises the memory space available at the receiver to the sender. The sender stores it in the so called send window.

A third buffer space limiting the throughput is the memory space available at the sender for a socket, called send buffer. Note that this is different from the send window, which relates to the memory available at the receiver.

The buffer has two different views, on one hand, the size expressed in TCP sequence numbers that relate to individual payload bytes, and on the other hand a memory view that reflects the true size of bytes used and reserved in memory for packet processing. Therefore, the true size in memory is much larger than the payload size. We explain this relation here in detail, since what usually is configured in operating systems are memory limitations on the size that can be consumed by a TCP socket. To understand the throughput limitation that is caused by memory limit configuration, it is essential to see how it translates to the sequence number space available. Hence, this limits the maximal data in flight. In the following, we adopt the notation as used in the Linux kernel.

Fig. 2 illustrates the structure of it in the view of sequence numbers as well as the memory. Sequence numbers are managed by the counters: `snd_una`, the lowest sequence number not acknowledged, `snd_nxt`, the sequence number used next, and `write_seq`, the next sequence number for application data. The range from `snd_una` to `snd_nxt` comprises data inflight but not acknowledged. The second range `snd_nxt` to `write_seq` contains backlogged data send from the application to the socket, but not yet sent to the network. For details see [17].

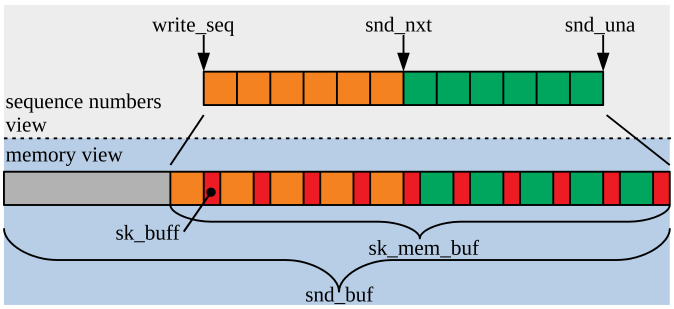

Fig. 2. Structure of the TCP send buffer and ensuing memory consumption. Each segment additionally consumes memory space for packet management, denoted by `sk_buff`.

The actual allocated memory is larger than just the payload. Each payload segment requires additional management data for packet processing, denoted as data structure `sk_buff`.

Throughout the paper, the configuration of buffer sizes refers to the memory size configuration and not the space of sequence number, since the used operating system only allows for the configuration of memory limits[1]

The values `snd_una`, `snd_nxt`, and `write_seq` indicate wheter the connection is application limited, receive or congestion window limited, or send buffer limited. If the application does not utilize the connection `snd_una` are `snd_nxt` are mostly equal. Only if the application generates data, the `snd_nxt` is greater if there are packets in flight, `snd_nxt` and `write_seq` are equal as long as packets can be sent directly. If the relation `write_seq` > `snd_nxt` > `snd_una` holds, the connection is either limited by the CWND or the receive window.

If the connection is limited by the send buffer size the relation `snd_una` $\ll$ `snd_nxt` = `write_seq` holds. The complete sequence number space is exhausted and `snd_nxt` as well as `write_seq` point to the end of the buffer. The blocking and unblocking of the socket is managed by the operation system by comparison of `sk_mem_queued` and `snd_buf` that translate to the true memory size as shown in Fig. 2. The operating system marks the socket as not writable, i.e. out of memory space, if `sk_mem_queued` approaches `snd_buf`. It is marked as writeable again, when the available space falls below a threshold. The buffer size in terms of sequence numbers can be read from `snd_nxt` - `snd_una`, which is completely consumed in this state.

In the experimental results presented later, we track `snd_una`, `snd_nxt`, `write_seq`, `snd_buf`, `sk_mem_queued` and available states as if the connection is CWND limited or the socket is not writable to evaluate the TCP scheduling behavior.

[1]We refer here to the memory limits configured in the Linux kernel with 'net.ipv4.tcp_wmem= *bmin bdefault bmax*' and 'net.ipv4.tcp_rmem= *bmin bdefault bmax*', respectively. An example, where the send buffer size `snd_buf` is dynamically adapted and grows to its maximum value *bmax*, is shown in Fig. 5.

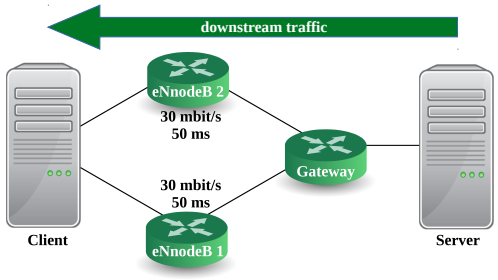

Fig. 3. Network setup with two paths from the client to the server, each path features a bandwidth of 30 Mbit/s and a delay of 100 ms. The queues at the routers, which are connected to the client, can hold up to 3000 packets. This is based on the fact that cellular networks provide also very deep queues.

B. Measurement Setup

For the evaluation of the root causes of the alternating utilization demonstrated in Fig. 1, we setup a simplified network topology as presented in Fig. 3. The network copies major characteristics of cellular networks to achieve a realistic performance degradation. The client is connected by two paths to the server, whereas each path corresponds to a single cellular connection. Both paths are aggregated and share the same path to the server. The bottleneck is configured at the routers (called eNodeB) on the paths to which the client is connected. Here, both paths are equally configured in the same Quality of Service (QoS) level with a bandwidth of 30 Mbit/s and a RTT of 100 ms. The queues can hold up to 3000 packets. This resembles the deep queues in cellular networks as measured in [14]. The effect of deep queues is also visible in Fig. 1. It is indicated by the massive increase of RTT in the range of multiple hundreds of milliseconds. For simplification a constant bandwidth and propagation delay is assumed here, to emphasize on the root causes, as we indicate in the following.

The network topology is built with Vagrant for automated testing using Kernel-based Virtual Machine. The virtual machines are based on Debian 9 and use the Linux Kernel 4.17 with MPTCP patches. The default minRTT scheduler is used, the path manager is set to full mesh, and all interfaces are configured as active. We note that in this basic network scenario the round-robin scheduler would be a preferred choice. Nevertheless, in this paper we stress on the relation of the minRTT scheduler, loss-based congestion control, and deep queues. Therefore, we limit our experiments to the default MPTCP scheduler based on minRTT, which is preferred in heterogeneous networks as evaluated in [10]. Traffic generation is performed with the software Iperf3 [18]. If not stated otherwise, TCP Reno is used, as congestion control, since the relation between throughput, queue sizes, RTT, buffer dimensioning is well-known.

C. Impact of Write and Read Memory

We first show the impact of read and write memory size available to a MPTCP socket and highlight the cases for low performance. We determine experimentally for a single path connection, that a send buffer of 1 MByte and a receive buffer

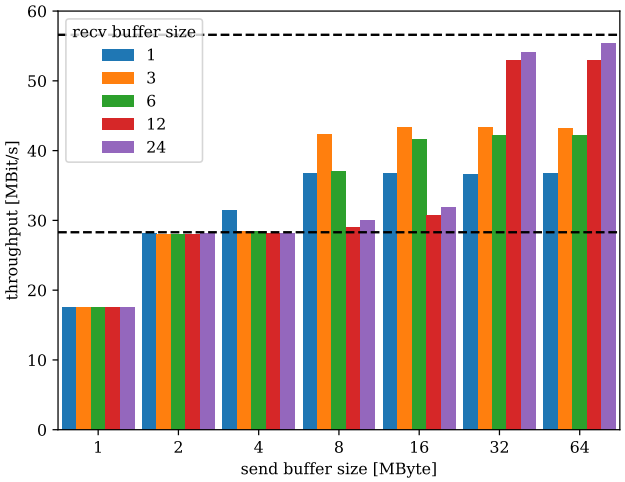

Fig. 4. Iperf3 throughput for MPCTP in dependence of send and receive buffer size. Two effects are visible: First, only for very large send and recive buffer sizes both flows are utilized, second, for send buffer sizes of 4 MByte and 8 MByte a smaller receiver window improves the throughput. The dashed lines indicate the maximal application layer throughput.

of 768 kByte are sufficient to fully utilize the path. These values are above the theoretical bandwidth delay product (BDP) of 100 $ms \cdot 30e6/8\ byte/s \approx 366\ kByte$, since the buffers account for the full memory utilization, which comprises data structures to manage packets and further packets waiting to be acknowledged as well as packets ready to be sent. The analysis of the difference between `snd_nxt-snd_una` shows a maximum of 384 kByte, which is close to the theoretical BDP. Since both paths have equal characteristics, a simple assumption is that doubling the buffer size leads to full utilization of both paths when MPTCP is used. But, Fig. 4 shows a different picture. For a utilization close to the theoretical maximum a send buffer of 32 MByte and a receive buffer of 12 MByte is required. The experiments were executed multiple times, the variance of results is typically small in the range of $\pm 5\%$.

A recommendation for buffer size dimensioning for MPTCP is also given in [16] to set the send and receive buffer to $2 \sum_{i=1}^{n} BW_i RTT_{max}$, where n is the number of paths, BW_i the bandwidth of path i and RTT_{max} the maximal RTT of all paths, i.e. in this use case the recommendation leads to 1.5 MByte. This value relates to the packet sizes without any overhead by the operating system. Still, the buffer size that achieves a full utilization is about ten times larger, which is not only due to overhead in the operating system.

Furthermore, the figure shows that only the throughput of one path is achieved up to a send buffer of 8 MByte. Three exceptions exists, namely, when the receiver window is small at send buffer size of 4 MByte and 8 MByte. Typically, larger buffers improve performance, here, the opposite effect occurs. We evaluate it in Sec. II-E. In particular both paths are fully utilized when the send and receive buffers are many times larger than required for full utilization of a single path connection.

We elaborate on these issues in the next section: Firstly, we explain why a send window needs to be multiple times greater than for single path TCP flows; secondly, we illustrate why in few cases a smaller receive window improves the performance.

D. Large send window

We analyze the requirements on the send window by reading status information from the MPTCP stack. To identify the requirements for a large send window, we show information on the CWND, the smoothed RTT (sRTT), and further check if the socket is CWND limited[2] or writable. CWND limited indicates that equal or more packets are in flight than allowed by the CWND, i.e. the link is fully utilized as estimated by the congestion control algorithm.

Fig. 5 shows the progress of these values for an exemplary MPTCP connection. We use TCP Reno as congestion control, send and receive buffer are set to default values of 4 MByte and 6 MByte, respectively. The default write buffer of Iperf3 is reduced from 128 kBytes to 1 kBytes. This reduction is for clarity, since a large write of 128 kBytes, which corresponds to about 92 packets, can only forward data to the network stack if sufficient space is available. The availability of sufficient space for large write buffers add an addition temporal behavior that increases the complexity of the figure.

In the beginning, the CWND increases according to slow start, also the sRTT increases from the minimal RTT of 100 ms to multiples of it. As indicated by packets in flight the subflows are used alternatingly, after a while, only subflow 2 is used, due to the smaller sRTT evaluated by the minRTT scheduler. The size of the sequence number space remains constant and the parameter *writeable* of subflow 2 often becomes false. This indicates that the socket buffer is limiting the maximum number of packets in flight.

As next, this state stabilizes. The sRTT of subflow 1 is not updated anymore, since it is not scheduled due to the outdated and higher sRTT, and subflow 2 cannot increase the sending rate further due to a limited send buffer. This creates a linkage between the queue size in network elements and send buffer size. Only if the send buffer is larger than the packets queued in the network, the second subflow is utilized as indicated by Fig. 4.

E. Small receive window

The results in Fig. 4 present contradictory values when the receive buffer size is decreased for a send buffer size of 4 MByte, 8 MByte, and 16 MByte, namely the throughput increases if the receive buffer size is decreased.

The difference in this case is that the receive buffer limits the throughput. Fig. 6 displays the value of the MPTCP socket that often decreased to small values, i.e. packets of larger size cannot be transferred. In our scenario, the data is forwarded from the application in chunks of 1 kByte.

In case of an exhausted receive window, the minRTT scheduler implements a receive buffer optimization that halves

[2]The implementation is in https://github.com/multipath-tcp/mptcp/blob/73bef74568c5238f68a693525401bfd6343e42ee/include/net/tcp.h#L1356

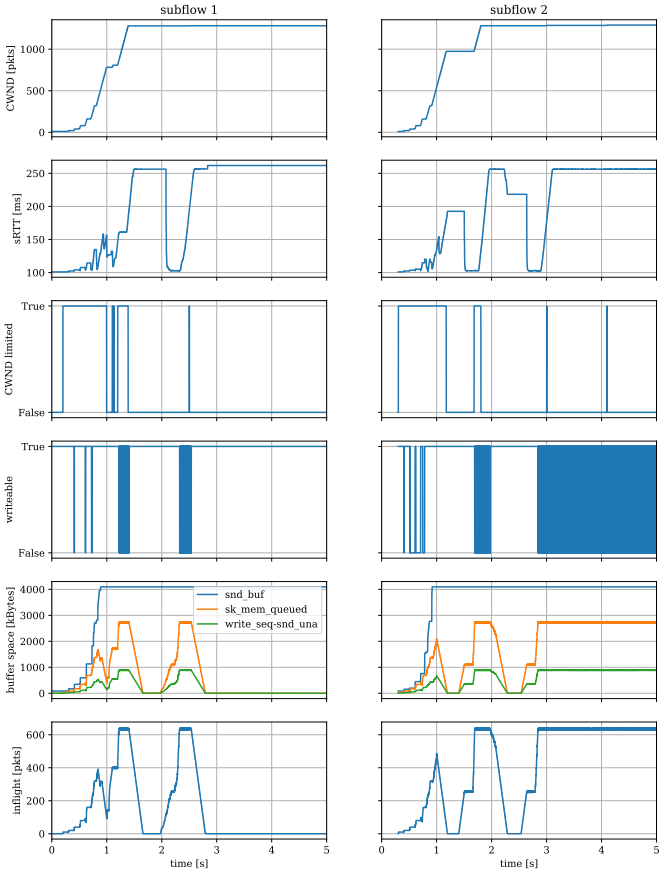

Fig. 5. TCP information of both subflows (top to down): CWND, sRTT, CWND limited, application socket is writeable, packets in flight. In the beginning both flows are utilized. CWND, sRTT, and packets in flight increase, sRTT increases significantly by self-induced congestion. At about 3 seconds, only subflow 2 is used to smaller RTT estimate, subfow 1 is not scheduled again and does not receive any RTT update. RTT of subfow 2 does not increase further since the send buffer space is utilized fully.

the window of non-selected subflows with greater RTT than the selected subflow. Due to the alternating reduction of the CWND, subflows become limited by the CWND. Thereby other subflows are selected, thus, the RTT is updated. This leads to an alternating usage of the flows, as shown by the CWND and sRTT values in Fig. 7.

III. Discussion of Mitigation

As described in the previous sections, multiple effects accumulate for poor performance and unnecessary buffer utilization: First, outdated RTT information, second, buffer requirements of loss-based congestion control, which further leads to significant increase of RTT by self-induced congestion.

We mitigate these effects by implementing a probing scheme, incorporating subflow availability based on the subflow send queue utilization, and by reduction of send buffer requirements using BBR as congestion control algorithm.

A. Tail Burst Probing

To address the outdated RTT information, we implement a probing mechanism that sends a TCP probe if the last

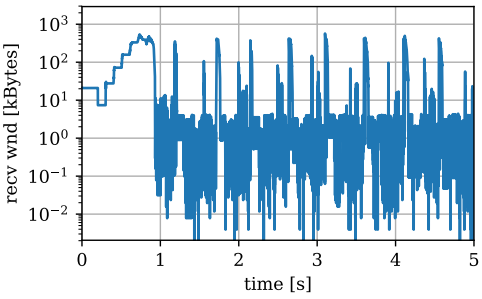

Fig. 6. Size of the receive window. Since the value decreases often to less than 1 kByte, the windows limits the throughput.

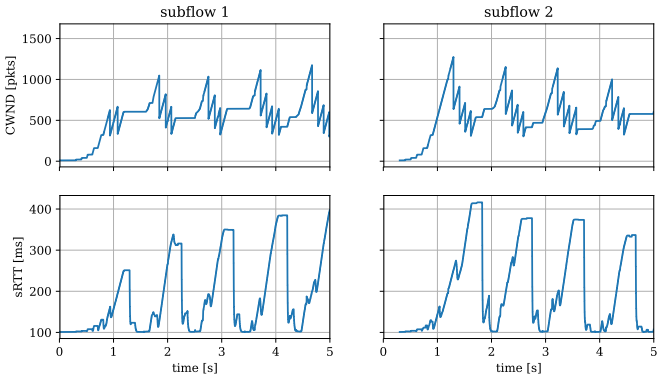

Fig. 7. CWND and sRTT for send buffer size of 8 MByte and receive buffer size of 3 MByte. With a limited receive window the subflows are used alternatingly

acknowledgement is older than the minimal RTT. To estimate the minimal RTT, we use the builtin estimator of the TCP stack. Furthermore, a probe packet is only sent once in the interval of the minimal RTT to avoid excessive probing.

Fig. 8 shows the benefit of such a probing. For the default setting of 4 MByte send buffer and 6 MByte receive buffer, the throughput increases above the capacity of a single link. Still, it is significantly below the maximal theoretical value. The limited performance is due to an alternating utilization of each subflows and thereby alternating consumption of the complete send buffer: A first subflow is utilized, the RTT increases and the send buffer space is consumed completely, a second subflow was utilized before and has a larger and outdated RTT estimation until it is probed. The duration between *a last packet is sent* and *the RTT update from probing* is here about 450 ms, which results from the RTT update from the last packet of the burst, about 250 ms, 100 ms of probing interval waiting time, and 100 ms of the minimal RTT propagation time.

This solution avoids starvation of subflows and increases performance, still bufferbloat and thereby significant increase of RTT persists as shown in Fig. 9. Also full utilization of both subflows is not achieved.

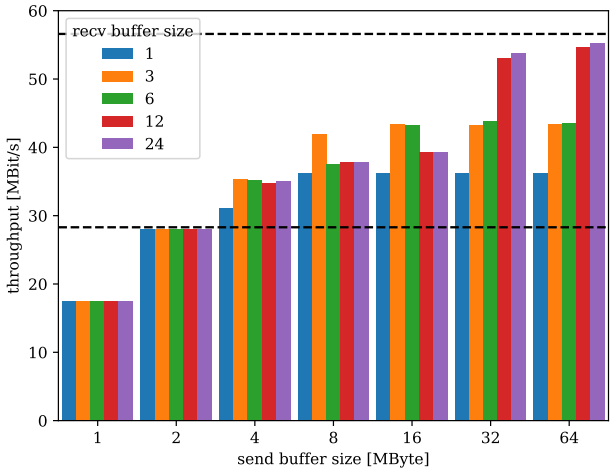

Fig. 8. Throughput for tail burst probing: The throughput for smaller buffer sizes increase if the RTT is updated by probing.

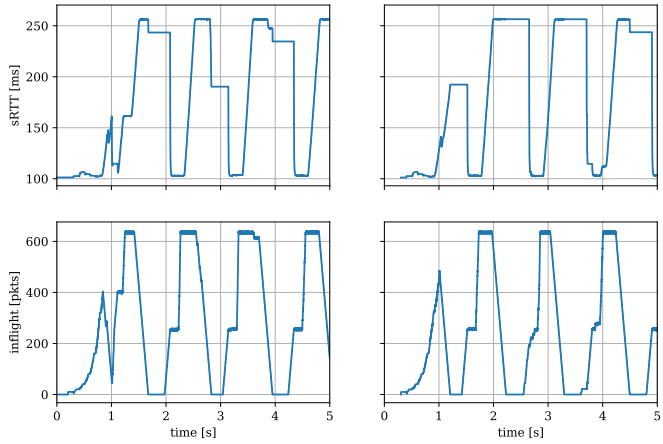

Fig. 9. Time series of sRTT and packets in flight when tail burst probing is used: Both subflows are used alternatingly, still the RTT increases significantly.

B. Tackling Bufferbloat

As previously described, a second reason for the poor performance is bufferbloat, which finally leads to starvation of one path. The problem is tackled in [19]. The authors consider a different scenario in which paths are heterogeneous in bandwidth and latency while the receive window limits the throughput. Since the origin of throttled performance is bufferbloat, we evaluate a similar solution in which the CWND is limited, when the smoothed RTT exceeds the minimal RTT of the path significantly. Divergent from the proposal in [19] and for ease of implementation, we use the minimal RTT estimation already present in the kernel implementation and reset the CWND to the calculated limit. According to the equation from [19] the CWND is set to:

$$CWND = \begin{cases} \lambda \frac{RTT_{min}}{sRTT} CWND, & \text{if } \frac{sRTT}{RTT_{min}} \geq \lambda \\ CWND, & \text{otherwise} \end{cases} \quad (1)$$

with $\lambda \in \{\frac{3}{2}, 3\}$.

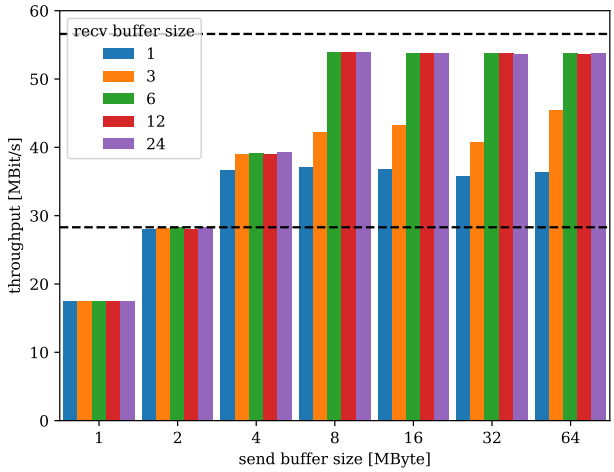

Fig. 10. Throughput when CWND is limited if bufferbloat is detected: The throughput improves if $\lambda = 3/2$ is used.

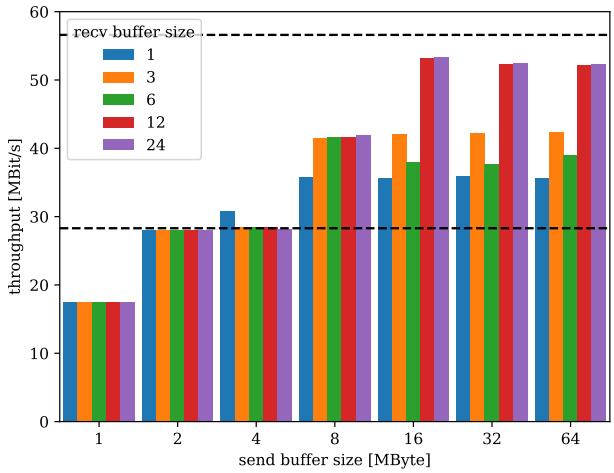

Fig. 11. Throughput when CWND is limited if bufferbloat is detected: The throughput does not improve if $\lambda = 3$ is used.

The results with parameters $\lambda \in \{\frac{3}{2}, 3\}$ are shown in the Fig. 10 and Fig. 11, respectively. For the $\lambda = 3$ no difference to the previous results in Fig. 4 are visible. Since the sRTT estimate in this scenario does not exceed 300 ms, limitation of the CWND does not apply. Reducing the parameter to $\lambda = \frac{3}{2}$ improves the performance as demonstrated in Fig. 10. Since often the relation $\frac{sRTT}{RTT_{min}}$ exceeds $\frac{3}{2}$. Still, this is not sufficient for a full utilization of both links. These results and the results in [11] demonstrate benefits of tackling bufferbloat, nevertheless, the results also highlight the challenge of parameter optimization. Furthermore, the limitation of the CWND is here implemented in the scope of the MPTCP scheduler, i.e., it is done outside of the congestion control algorithm implementation. Next, we address this last issue, by letting the congestion control algorithm tackling bufferbloat.

C. Bottleneck Bandwidth and Round-trip Time (BBR)

The previous section indicates that the reduction of bufferbloat increases the overall throughput performance. In

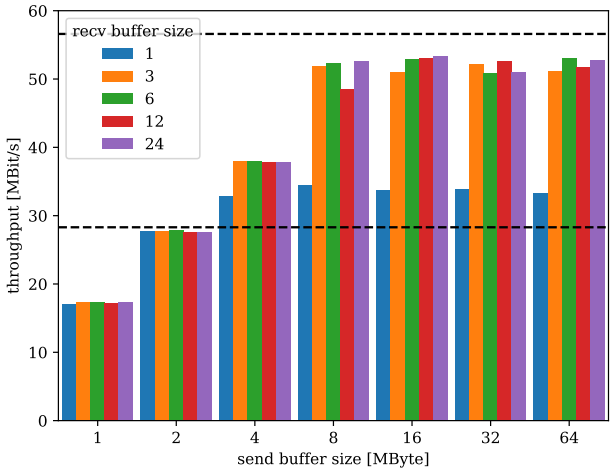

Fig. 12. Throughput for send and receive buffer configuration and congestion control algorithm BBR: Already for the default configuration of 4 MByte send buffer and 6 MByte receive buffer, the throughput increases above the single path throughput and increases further with larger buffers.

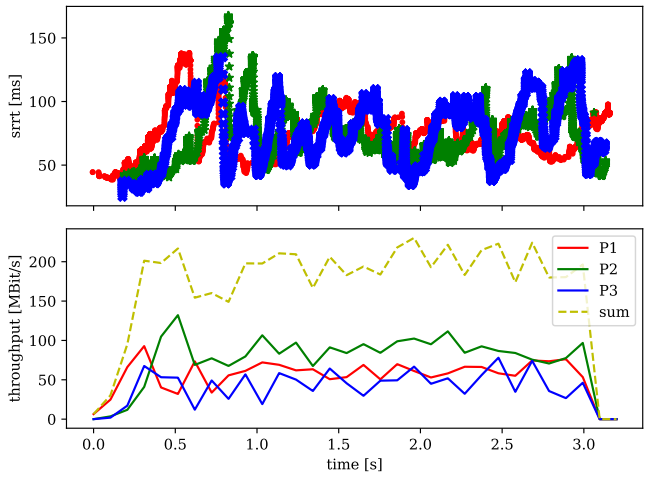

Fig. 13. The use of BBR as congestion control protocol for MPTCP leads to a reduction of sRTT and throughput increase. No drops in throughput and excessive delay increases occur compared to the introductory real-world experiment using Reno instead of BBR.

the previous section, the bufferbloat reduction is performed by limitation of the CWND during scheduling. The optimization goal of utilizing a link without the creation of extensive queueing is implemented by the congestion control algorithm BBR [20]. Therefore, this congestion control algorithm addresses bufferbloat, too.

The results in Fig. 12 show the throughput using BBR as congestion control algorithm, which improves the throughput compared to TCP Reno. Since BBR reduces bufferbloat and is not sensitive to parameter optimization, as the two previous approaches, it is promising for combination with minRTT scheduling. We deploy TCP BBR in our real-world experimental setup presented in Sec. I. The sRTT and throughput results are presented in Fig. 13. The sRTT still increases but much less than in the introductory results shown in Fig. 1 and moreover the throughput is much more balanced, too.

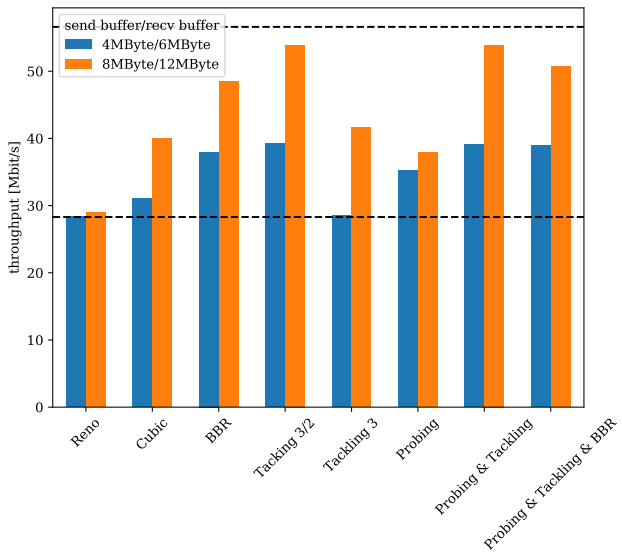

Fig. 14. Direct comparison of throughput for congestion control algorithms and mitigation techniques. All mitigation techniques we present improve the throughput. As discussed before BBR is promising due to reduction of RTT and no need for parametrization.

Even though the results are promising, the BBR algorithm is not a panacea due to issues on fairness as reported in [21], [22]. Still the results demonstrate that minRTT scheduling requires a congestion control algorithm that avoids bufferbloat on the paths, since bufferbloat distorts the scheduling metric.

Finally, we show a comparison for the default buffer sizes of send and receive buffer in Fig. 14. Additionally to the previous results, throughput measurements for the congestion control Cubic [23] and combinations of the presented mitigation techniques are included. Fig. 14 indicates most increase of throughput for using the tackling approach with parameter $\frac{3}{2}$ and the BBR approach. Both approaches avoid bufferbloat by adaptation of the CWND. Also the tackling approach with parameter $\lambda = 3$ shows an improvement for larger buffer sizes. Due to large buffer sizes the RTT increases and the sRTT increases above $3 \cdot$minRTT, so the CWND is adjusted also with parameter $\lambda = 3$, too. The combination of the approches does not show an further improvement. Still, we think BBR is most promising since it avoids the conflict of minRTT scheduling and loss-based congestion control algorithms, that typically increase the RTT by their greedy behavior and avoid complex parameter optimization.

IV. Related Work

In this paper, we address the problem of buffer dimensioning for multi-path protocols. Especially, the problem is mostly addressed in the scope of the receiver buffer and the scheduler of MPTCP. We shed light on a limitation according to send buffer limitation, still mitigation techniques for reduction of bufferbloat improve the performance independent from the send or receive buffer.

In [10], different scheduling disciplines where compared that finally lead to the current default scheduler of MPTCP,

namely the minRTT scheduler, which prefers the path with the smallest RTT as long as the path is not overloaded. Overload detection is done by congestion control protocols taken from single path TCP or newly designed protocols as LIA [6], OLIA [7], BALIA [8], and wVegas [9] that address the problem for shared bottlenecks. We did not test such protocols since for our experiments the bottlenecks are disjoint.

An active probing approach to avoid outdated RTT information is also presented in [24]. The authors in [24] address by their approach thin streams, whereas we show that outdated RTT information also occurs for greedy traffic flows. For thin streams they recommend much higher probing interval. In both cases active probing increases the performance.

HoL blocking issues are addressed in [11], [12], [13]. In [12], a schedule for packet transmission times is generated, whereas in [13] the scheduler calculates the arrival time for the packets ready to be sent to achieve in-order arrival. Both approaches are compared in [11] and additionally the authors [11] introduce a new scheduling algorithm that avoids the utilization of flows with the larger RTT if the data would cause HoL at the receiver. All algorithms optimize on the problem of HoL blocking at the receiver side. Still, the issue of large RTT due bufferbloat in network queues remains and is not addressed directly by these approaches. Nevertheless, addressing HoL blocking also reduces the memory allocation at the sender, which mitigates the effects we observe. A scheduler designed for reduction of delay is presented in [25]. The authors further point out that in heterogeneous network paths the redundant scheduler often sends outdated packets on the path with small bandwidth and low latency. This behavior impedes advantages by redundant scheduling since effectively only packets of one path are usable.

V. Conclusion

Scheduling in multi-path protocols is challenging, it has to account for heterogeneous network paths and resulting effects as e.g. HoL blocking. In this paper, we contribute insights to the issue of bottlenecks with deep queues in combination with multi-path protocols. Such path characteristics are prevalent for cellular networks. We show the interaction of the default scheduler of MPTCP, which uses the minimal RTT as decision criteria, and loss-based congestion control protocols. This causes extensive bufferbloat at network elements and so distorts the RTT measurements of network paths significantly. This distortion leads to poor performance of MPTCP in such scenarios due to outdated RTT measurements and starvation of subflows. We present mitigation techniques by implementing a probing scheme and avoidance of extensive queueing on the network path. Reducing bufferbloat by selection of a non loss-based congestion control algorithm improves performance in laboratory as well as real-world experiments. The experiments reveal the shortcomings of loss-based congestion control protocols, since loss in current cellular network is small, and reveal the need for sophisticated congestion control protocols for multi-path protocols.

References

[1] T. Viernickel, A. Froemmgen, A. Rizk, B. Koldehofe, and R. Steinmetz, "Multipath QUIC: A Deployable Multipath Transport Protocol," in *IEEE ICC*, May 2018.

[2] Q. D. Coninck and O. Bonaventure, "Multipath QUIC: Design and Evaluation," in *Proc. of ACM Conext*, Dec. 2017.

[3] R. R. Stewart, "Stream Control Transmission Protocol," RFC 4960, Sep. 2007.

[4] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 951–964, Oct. 2006.

[5] C. Paasch and O. Bonaventure, "Multipath TCP," *ACM Queue*, vol. 12, no. 2, pp. 40:40–40:51, Feb. 2014.

[6] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, Implementation and Evaluation of Congestion Control for Multipath TCP," in *Proc. of USENIX NSDI*, ser. Proc. of NSDI, Mar. 2011, pp. 99–112.

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

[8] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath TCP: Analysis, Design, and Implementation," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 596–609, Feb. 2016.

[9] Y. Cao, M. Xu, and X. Fu, "Delay-based congestion control for multipath TCP," in *Proc. of ICNP*, Oct. 2012.

[10] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental Evaluation of Multipath TCP Schedulers," in *Proc. of the ACM SIGCOMM Workshop on CSWS*. ACM, 2014, pp. 27–32.

[11] S. Ferlin, O. Alay, O. Mehani, and R. Boreli, "Blest: Blocking estimation-based mptcp scheduler for heterogeneous networks," in *Proc. of IFIP Networking*, May 2016, pp. 431–439.

[12] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent Delay-aware Packet Scheduling for Multipath Transport," in *Proc. of IEEE ICC*, Jun. 2014, pp. 1222–1227.

[13] F. Yang, Q. Wang, and P. D. Amer, "Out-of-Order Transmission for In-Order Arrival Scheduling for Multipath TCP," in *Proc. of Conference on Advanced Information Networking and Applications Workshops*, May 2014, pp. 749–752.

[14] N. Becker, A. Rizk, and M. Fidler, "A Measurement Study on the Application-level Performance of LTE," in *Proc. of IFIP Networking*, Jun. 2014.

[15] X. de Foy, M. Perras, U. Chunduri, K. Nguyen, M. G. Kibria, K. Ishizu, and F. Kojima, "Considerations for MPTCP Operation in 5G," Internet-Draft, Jun. 2011.

[16] J. Iyengar, C. Raiciu, S. Barre, M. J. Handley, and A. Ford, "Architectural Guidelines for Multipath TCP Development," RFC 6182, Mar. 2011. [Online]. Available: https://rfc-editor.org/rfc/rfc6182.txt

[17] A. Agache and C. Raiciu, "Oh Flow, Are Thou Happy? TCP Sendbuffer Advertising for Make Benefit of Clouds and Tenants," in *Proc. USENIX Workshop HotCloud*, Jun. 2015.

[18] "Iperf3," https://iperf.fr/iperf-download.php.

[19] S. Ferlin-Oliveira, T. Dreibholz, and O. Alay, "Tackling the Challenge of Bufferbloat in Multi-Path Transport over Heterogeneous Wireless Networks," in *Proc of. IEEE IWQoS*, May 2014, pp. 123–128.

[20] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-Based Congestion Control," *ACM Queue*, vol. 14, September-October, pp. 20 – 53, 2016.

[21] D. Scholz, B. Jaeger, L. Schwaighofer, D. Raumer, F. Geyer, and G. Carle, "Towards a Deeper Understanding of TCP BBR Congestion Control," in *Proc. of IFIP Networking*, May 2018.

[22] M. Hock, R. Bless, and M. Zitterbart, "Experimental Evaluation of BBR Congestion Control," in *Proc. of IEEE ICNP*, Oct. 2017.

[23] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-friendly High-speed TCP Variant," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[24] A. Froemmgen, J. Heuschkel, and B. Koldehofe, "Multipath TCP Scheduling for Thin Streams: Active Probing and One-Way Delay-Awareness," in *In Proc. of IEEE ICC*, May 2018.

[25] B. Walker, V. A. Vu, and M. Fidler, "Multi-Headed MPTCP Schedulers to Control Latency in Long-Fat / Short-Skinny Heterogeneous Networks," in *Proc. of ACM CHANTS*, 2018, pp. 47–54.