Low-Latency Scheduling in MPTCP

Per Hurtig, *Member, IEEE*, Karl-Johan Grinnemo, *Senior Member, IEEE*, Anna Brunstrom, *Member, IEEE*, Simone Ferlin, Özgü Alay, and Nicolas Kuhn

***Abstract*— The demand for mobile communication is continuously increasing, and mobile devices are now the communication device of choice for many people. To guarantee connectivity and performance, mobile devices are typically equipped with multiple interfaces. To this end, exploiting multiple available interfaces is also a crucial aspect of the upcoming 5G standard for reducing costs, easing network management, and providing a good user experience. Multi-path protocols, such as multi-path TCP (MPTCP), can be used to provide performance optimization through load-balancing and resilience to coverage drops and link failures, however, they do not automatically guarantee better performance. For instance, low-latency communication has been proven hard to achieve when a device has network interfaces with asymmetric capacity and delay (e.g., LTE and WLAN). For multi-path communication, the data scheduler is vital to provide low latency, since it decides over which network interface to send individual data segments. In this paper, we focus on the MPTCP scheduler with the goal of providing a good user experience for latency-sensitive applications when interface quality is asymmetric. After an initial assessment of existing scheduling algorithms, we present two novel scheduling techniques: the block estimation (BLEST) scheduler and the shortest transmission time first (STTF) scheduler. BLEST and STTF are compared with existing schedulers in both emulated and real-world environments and are shown to reduce web object transmission times with up to 51% and provide 45% faster communication for interactive applications, compared with MPTCP's default scheduler.**

***Index Terms*— Transport protocols, MPTCP, low-latency, scheduling, asymmetric paths.**

I. Introduction

TODAY, the number of mobile devices exceeds the world's population, according to Cisco's Global Mobile Data Traffic Forecast [1]. Trends suggest that mobile devices are becoming end users' de facto communication device in part thanks to technology advancements that enable employing multiple interfaces for communication, e.g. LTE and WLAN. The use of multiple interfaces has made Internet access truly ubiquitous as users now can access online services at home, while commuting, or even when hiking.

P. Hurtig, K.-J. Grinnemo, and A. Brunstrom are with the Department of Computer Science, Karlstad University, 651 88 Karlstad, Sweden (e-mail: per.hurtig@kau.se; karl-johan.grinnemo@kau.se; anna.brunstrom@kau.se).

S. Ferlin is with Ericsson Research, 164 40 Stockholm, Sweden (e-mail: simone.ferlin@ericsson.com).

Ö. Alay is with the Simula Research Laboratory, 1364 Oslo, Norway (e-mail: ozgu@simula.no).

N. Kuhn is with the French National Space Centre, Centre National d'Etudes Spatiales, 31 401 Toulouse, France (e-mail: nicolas.kuhn@cnes.fr).

Digital Object Identifier 10.1109/TNET.2018.2884791

With more devices, the demand for data transmission within mobile networks has also increased significantly, requiring new technical solutions and innovative network designs. Operators providing both LTE and WLAN access see this as an opportunity to provide simultaneous access, simplifying mobility and network off-loading. Aggregating LTE and WLAN access at the data-link layer can result in improved performance [2]. However, deployment of such solutions is complex since operators have to consider already deployed networks, each of them having its specific Medium Access Control (MAC) management. On the other hand, Multi-path TCP (MPTCP) [3] emerged as a drop-in extension to TCP and can be deployed as a proxy before the aggregation networks of both LTE and WLAN networks. Despite being a recent standard, MPTCP is already widely deployed and used by major software vendors, including Apple and its Siri service [4], [5].

Recently, a lot of research and development have exploited MPTCP's multi-path transmission capabilities to offer new transport services and performance enhancements. For example, MPTCP can be used to provide seamless handover between network interfaces to enable user mobility and resilience to link failures [6]. Performance improvements, compared to single-path protocols, include, e.g. simultaneous use of multiple paths for capacity aggregation [7]. Aggregating capacity can be useful in, e.g. data center scenarios [8], where nodes often need to transmit a large amount of data and have a high degree of inter-connectivity. Capacity aggregation does, however, not always work well when asymmetric link technologies are used (e.g. LTE and WLAN) and low latency is more important than high throughput [9]. The lack of proper support for low-latency capacity aggregation is a serious drawback for many applications. For example, consider an end user that is trying to chat with a friend using a messaging service on a smartphone. For such applications, throughput does not matter, since the users just want to have a smooth and responsive communication with minimum latency.

This article focuses on low-latency capacity aggregation, by considering different scheduling techniques in MPTCP. First, the limitations of state-of-the-art MPTCP schedulers are explored and analysed. Most schedulers are found to underperform with asymmetric paths, where they are unable to provide low latency and utilise the aggregate capacity. Then, two novel schedulers are proposed to address shortcomings related to asymmetry: BLock ESTimation (BLEST) scheduler [10] which aims to reduce buffer blocking and Shortest Transfer Time First (STTF) [11] that tries to minimise the transmission time of each data segment. The design of BLEST and STTF is influenced by the limitations of previously proposed schedulers. While BLEST is a lightweight solution, STTF is more complex and requires more computational resources. The article covers both the design and the implementation of these scheduling techniques in the Linux kernel. The schedulers are also evaluated using realistic application scenarios in both emulations and real-world environments. The results show that both BLEST and STTF fulfil their design goals and can reduce application latency significantly. Compared to MPTCP's default scheduler, experiments show that STTF can reduce web object transmission times with up to 51% and provide 45% faster communication for interactive applications. We further provide the open source implementation along with all the necessary changes to MPTCP for both BLEST and STTF to the community [12], [13].

The rest of the article is structured as follows. Section II gives the necessary background and motivation for this work, including an overview of MPTCP and the general problem of communicating over asymmetric paths efficiently. Section III presents the state-of-the-art in MPTCP scheduling; schedulers that aim to perform well under both symmetric and asymmetric conditions. Although some of the schedulers presented in Section III work quite well, they are still unable to deliver low-latency communication. Section IV therefore introduces and describes BLEST and STTF – two novel schedulers designed for asymmetric interface usage. In Section V, BLEST and STTF are evaluated through basic latency and throughput experiments, as well as emulated network scenarios involving both web traffic and traffic generated by a typical interactive application, Google Maps. The final step of the evaluation is given in Section VI, where the schedulers are evaluated in a series of real-world web experiments. The article ends with related work in Section VII, and a summary and avenues for future work in Section VIII.

II. Background and Motivation

This section shortly describes the basics of MPTCP, the general problems of efficiently transmitting data over asymmetric paths, and how MPTCP is unable to resolve these challenges, making it unsuitable for applications with low-latency requirements.

A. The Multi-Path TCP (MPTCP) Protocol

To allow for cheap and ubiquitous Internet access, smartphone providers are equipping devices with multiple network interfaces. The reason for this has been to enable handover from slow and expensive mobile data networks to fast and free WLANs, when possible. This is now changing as mobile data networks are becoming faster (e.g. the upcoming 5G standard) and cheaper. Therefore, the use case of multi-access devices is no longer limited to flipping between radio technologies, it is now possible to use multiple technologies simultaneously to increase throughput and robustness against network path failures [14]. MPTCP was designed as a set of transport extensions to TCP, allowing already existing applications to benefit from such interface aggregation. To date, MPTCP is standardised by the IETF in RFC 6824 [3], and a de facto state-of-the-art reference implementation exists for Linux [15]. Apart from the reference implementation, there are several commercial and non-commercial options available [16], including Apple's iOS implementation [5].

Figure 1 shows how MPTCP is situated in the network stack. An MPTCP connection is comprised of one or more subflows which typically are bound to separate network interfaces. For example, one subflow over an LTE interface and one over a WLAN interface. Subflows are regular TCP connections and MPTCP's main task is to transparently split data among the subflows. When an MPTCP connection contains more than

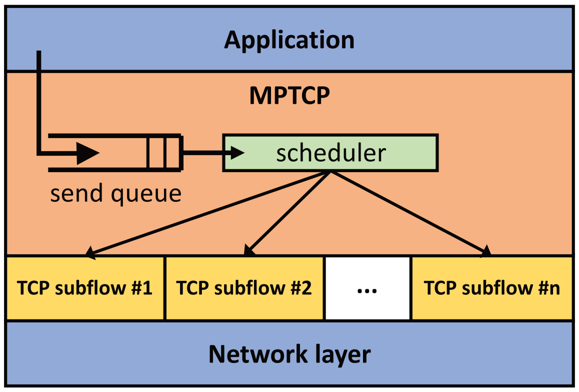


Fig. 1. MPTCP connections contains several TCP subflows which are employed to send data according to the scheduling policy of the selected scheduler.

one subflow the data scheduler decides, for each segment, which subflow to use for transmission. The decision is typically done with respect to network properties of the subflows' underlying network paths. For instance, if a particular subflow has a very low round-trip time (RTT) compared to the other available subflows it might be beneficial to send data on that subflow. The task of efficiently scheduling data over paths with different characteristics is hard, especially if their capacity and/or latency differs significantly.

B. Communication Over Asymmetric Paths

Most Internet-based applications require that data is received in the same order it was sent. Since multi-path protocols like MPTCP split data among several paths, chances are that data arrive out-of-order. This is especially true if paths are asymmetric, in terms of capacity and/or delay. When data arrives out-of-order a number of performance-related problems may occur, all well-known problems in transport-layer research [17]. Figures 2a to 2d illustrate two inherent problems when transmitting data over paths with asymmetric characteristics (in this example RTTs). Figure 2a illustrates two end-hosts, where the leftmost is about to send a number of data segments to the host on the right-hand side using two paths. As shown in the figure, the upper path has an RTT that is ten times larger than the RTT of the lower path. The sending host starts transmitting segments 1 and 2 over the slow path and segments 3 and 4 over the faster path, as shown in Figure 2b. For this example, Figure 2c shows the well-known Head-of-Line (HoL) blocking problem, where segments 3 and 4 have been received and are kept buffered until segments 1 and 2 arrive, potentially causing delays. A problem related to HoL blocking is receiver buffer blocking. While HoL blocking introduces a delay in delivering data to the application, it does not interact with the sending of data. Receiver buffer blocking, however, reduces throughput and can, therefore, further increase delay. This is illustrated in Figure 2d; if more data is sent over the faster subflow, the receiver's buffer might be filled with data that cannot be delivered until segments 1 and 2 have been received. When the receiver's buffer becomes completely filled, the sender will be blocked from sending, causing both reduced throughput and increased latency.

To avoid these problems, multi-path protocols need to carefully select which interface to use for transmission of each individual segment. The employed scheduling strategy must

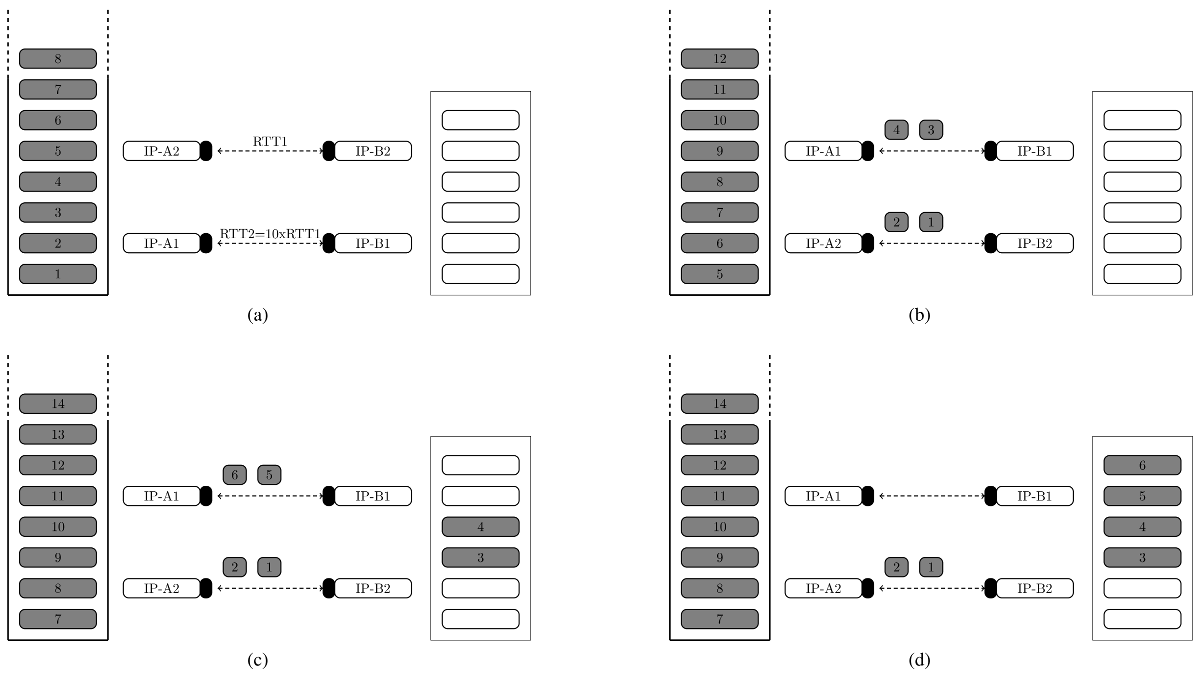


Fig. 2. Illustration of Head-of-Line Blocking and Receiver Buffer Blocking. (a) At t_0; asymmetric path $RTT_2 = 10 \times RTT_1$. (b) At $t_1 = t_0 + \epsilon$; #1 and #2 on path 1; #3 and #4 on path 2. (c) At $t_2 = t_1 + RTT_1$; #3 and #4 are received but can not be read because #1 and #2 are missing $\Rightarrow$ Head-of-Line (HoL) blocking. (d) At $t_3 = t_1 + 2 \times RTT_1$; #1 and #2 are not received and the receiver's buffer can not receive #7 $\Rightarrow$ receiver buffer blocking.

avoid overuse of slower network paths, as they will limit the aggregate performance of the connection.

C. MPTCP Over Asymmetric Paths

For the work described in this article, the Linux kernel MPTCP implementation is used. Although no default scheduler has been standardised for MPTCP, most implementations, including the Linux implementation, use some variant of the Lowest-RTT-First (LRF) scheduler [18]. LRF was designed to maximise throughput by prioritising subflows with low RTTs: for all subflows with available space in their CWNDs, schedule segments over the one with the lowest smoothed RTT (SRTT).

While this strategy effectively maximises connection throughput when network paths are symmetric, it works poorly for asymmetric paths. It is rather surprising that new protocols like MPTCP does not have a good strategy for communicating over asymmetric paths, especially since one of MPTCP's core use cases [16] involves network technologies that are inherently asymmetric (i.e. WLAN and cellular). Actually, a simple switch from TCP (running over WLAN) to MPTCP (running over both WLAN and 3G) could ruin the performance of a simple file transfer. This is illustrated in Figure 3, which shows the transmission times of a number of file transfers. The transfers were of different sizes (10-100 segments large) and were conducted over real WLAN and 3G networks running the Linux kernel implementations of TCP and MPTCP. As hinted by the graph, MPTCP is unable to properly deal with the asymmetry between the two available interfaces which causes the slower 3G path to limit the performance.

The simple explanation for MPTCP's performance problems is that slow subflows tend to be used when they really should not. The reasons for such overuse are actually many, present in protocol and scheduler design as well as in unexpected interactions between MPTCP and TCP. For example, one of

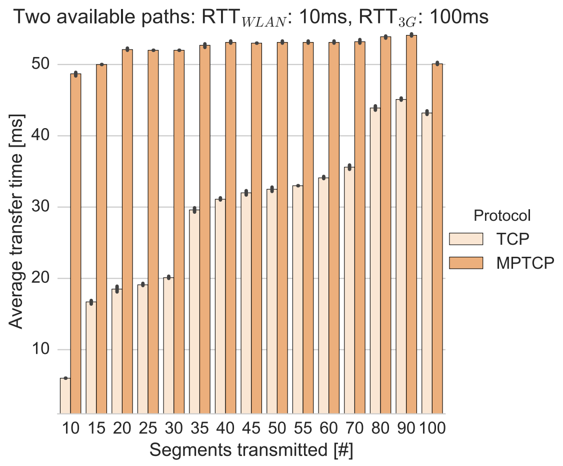


Fig. 3. Transmission time of differently sized files, using TCP and MPTCP, with two available paths having asymmetric RTTs (10 ms vs 100 ms).

LRF's problems is the strict use of the CWND to select which subflows that are suitable for scheduling data on. Let us elaborate on what happens when 15 segments are to be transmitted over an MPTCP connection with one subflow routed over a WLAN (with a 10 ms RTT) and the other over 3G (with a 100 ms RTT). LRF will immediately schedule segments on the WLAN subflow, as it has the lowest RTT. However, if the MPTCP connection has just been opened, or idle for a short while, the initial CWND size will limit the number of segments to ten. The remaining five segments will have to be scheduled on the 3G subflow. LRF causes the transfer of 15 segments to last for at least 50 ms ($\frac{\text{RTT}_{3G}}{2}$) as each subflow will be utilised in parallel and the slow subflow dominates. (We disregard the bandwidth in this example.) TCP,

TABLE I
MAIN PROPERTIES AND KEY METRICS OF THE LRF, DAPS, OTIAS AND ECF SCHEDULERS

Algorithm	Overall Goal	Input Metrics	Remarks	Approach
LRF [18]	Maximise throughput	CWND, RTT	Takes lowest SRTT and CWND space to prioritise and use all available subflows.	Reactive
DAPS [20]	Efficient use of the receive buffer, translating into reduced HoL-blocking.	CWND, RTT	Takes lowest SRTT and CWND space to prioritise and use all available subflows.	Reactive
OTIAS [21]	Shortest time to arrive at the receiver regardless if the subflow(s) has space in its CWND.	RTT	Takes lowest SRTT and CWND to prioritise, eventually skipping subflow(s).	Proactive
ECF [22]	Minimise idle periods of the fast subflow(s) waiting for slow subflow(s).	CWND, RTT and buffered data to be sent	Takes lowest SRTT first, eventually skipping some subflow(s).	Proactive

on the other hand, only utilises the WLAN and is finished in $10+5=15\,\text{ms}$, as one and a half RTT is required to complete the transfer.

In addition to scheduler design issues, unexpected protocol interactions between TCP and MPTCP can occur as TCP is optimised for single-path use. Given that the work described in this article used the Linux kernel implementation of MPTCP, a few interaction problems between MPTCP and TCP in Linux are given as examples. One such example is Linux's use of stale RTT measurements, which may lead to the abandonment of a fast subflow in favor of a slower one. Consider two subflows, S_1 and S_2, which experience RTTs of RTT_{S_1} and RTT_{S_2} where $RTT_{S_1} < RTT_{S_2}$. Since S_1 has the lowest RTT, it will be the preferred subflow. (In this example, we assume an equal capacity for the two subflows.) Now, let's assume that S_1 starts to fill up intermediate queues, and, as a result, RTT_{S_1} starts to increase. At the time when RTT_{S_1} becomes larger than RTT_{S_2}, MPTCP will start using S_2. If we further assume that RTT_{S_1} later on starts to decrease and that we eventually fall back to the initial situation with $RTT_{S_1} < RTT_{S_2}$, MPTCP will not switch back to S_1: Since MPTCP stopped sending data over S_1 when $RTT_{S_1} > RTT_{S_2}$, it is not aware of the decrease in RTT_{S_1}. Additional examples of unexpected implementation interactions include the TCP Small Queues (TSQ) mechanism [19]. TSQ is designed to reduce latency among concurrent flows by limiting the amount of data that can be queued within the TCP stack. Currently, its default limit is set to the minimum of two data segments or $1\,\text{ms}$ worth of data. When segments are about to be scheduled on a subflow, the scheduler first checks whether TSQ indicates that its limit has been reached. In most situations, this limit is reached rather quickly, after a few segments, with the result that MPTCP abandons the current subflow and chooses the next in line. The consequence of this behaviour is that MPTCP may switch to a slower subflow prematurely; long before the CWND of the faster subflow has been filled. This is illustrated in Figure 3, where the ten-segment transfer for MPTCP clearly utilises both paths, although the congestion window supports sending all data over the faster path.

To avoid buffer blocking problems, due to overuse of slow subflows, a number of schedulers for MPTCP have been proposed. The following section will examine the three state-of-the-art schedulers DAPS, OTIAS, and ECF, and compare their performance to the default LRF scheduler.

III. PERFORMANCE EVALUATION OF THE STATE-OF-THE-ART MPTCP SCHEDULERS

As described earlier, MPTCP's default LRF scheduler operates by filling the congestion window (CWND) of the subflow with the lowest RTT before advancing to other subflows in ascending RTT order. This approach works well when using symmetric paths, but has otherwise been proven suboptimal. In this section, we analyse three state-of-the-art MPTCP schedulers designed to overcome LRF's problems with asymmetric path conditions: DAPS, OTIAS, and ECF. An overview of the properties of these three schedulers along with the baseline LRF is given in Table I. Next, we detail these schedulers, compare their performance under symmetric and asymmetric paths, and finally discuss their strengths and shortcomings.

A. Delay-Aware Packet Scheduler (DAPS)

The main idea behind the DAPS scheduler [20] to schedule the segments over the available subflows in such a way that the probability of in-order reception at the destination is maximised. In that sense, the main objective is not to reduce latency per se, but to ascertain the efficient use of the MPTCP receive buffer at the destination, and in so doing minimise the chances of receive-buffer blocking.

The DAPS algorithm comprises of three phases. In the first phase, the order in which the available subflows should be employed for transmission of segments is determined. On the basis of this order, the second phase involves determining which segments to send over each available subflow every time the DAPS scheduler is invoked. Finally, in the third and last phase, segments are sent in accordance with the determined schedule in phase 2.

B. Out-of-Order Transmission for In-Order Arrival Scheduler (OTIAS)

The OTIAS scheduler [21] builds on the idea of scheduling a segment on the subflow with the shortest transfer time, i.e. on the subflow over which it takes the shortest time to arrive at the destination. Here, OTIAS includes a simplification, assuming that the one-way delay is RTT/2. This means that the OTIAS scheduler differs from LRF in that it permits the scheduling of segments on all available subflows, not only those with available CWND space.

Every time a segment is to be scheduled, the OTIAS scheduler computes the expected transfer time over each of the available subflows. The computation of the transfer time on those subflows with no available CWND space entails calculating the number of RTTs worth of data that is queued on these subflows. Eventually, the segment is scheduled on the subflow with the shortest expected transfer time.

C. Earliest Completion First (ECF)

The ECF scheduler aims to tackle the performance degradation problems that arise from path heterogeneity [22].

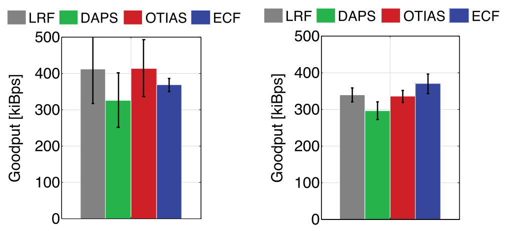


Fig. 4. Goodput for bulk traffic: LRF, DAPS, OTIAS, and ECF. (a) **WLAN/3G**. (b) **WLAN/WLAN**.

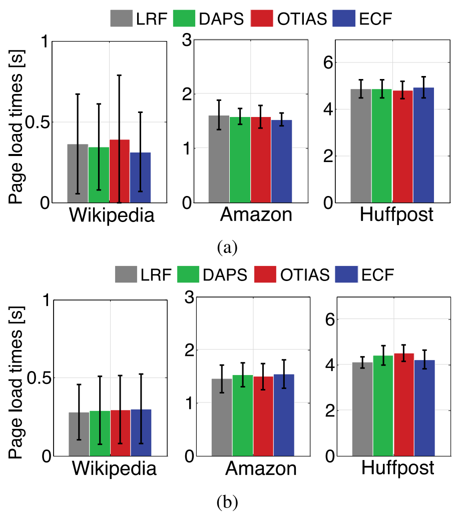


Fig. 5. Page load time for web traffic (Wikipedia, Amazon and Huffington Post) for LRF, DAPS, OTIAS and ECF. (a) **WLAN/3G**. (b) **WLAN/WLAN**.

More specifically, ECF aims to minimise the periods where a fast subflow becomes idle, i.e. waiting until a slow subflow completes its assigned packet transmissions, due to the lack of packets to send.

In order to make a scheduling decision, ECF not only monitors the subflows' RTT estimates, but also the size of the CWND and the amount of data available to send. More concretely, ECF takes the following decisions: ECF always prefers the fastest subflow, i.e. the subflow with the shortest RTT, as long as it is able to send on that subflow. Otherwise, ECF searches for the next fastest subflow, i.e. the next subflow with the shortest RTT, which also has space in its CWND. To take the decision, whether to wait for the fastest subflow or use another one, ECF calculates the arrival time for a segment that can be sent now on both subflows. If the required time is at least twice the RTT of the fastest subflow, ECF waits for the fastest subflow instead.

D. Performance Evaluation and Discussion

Next, we evaluate the performance of LRF, DAPS, OTIAS, and ECF using an emulated network with a non-shared bottleneck topology. The settings and further experimental configuration are detailed in [10].

In Figure 4, we illustrate the performance of the schedulers using bulk transfer as the target application and application goodput as our metric. For the WLAN/3G setting, we observe that OTIAS provides similar performance to LRF, while DAPS and ECF perform worse than LRF. For the WLAN/WLAN case, DAPS performs worse than LRF while OTIAS having virtually the same goodput as LRF. However, we observe that ECF provides gains compared to LRF.

In Figure 5, we evaluate the performance of the schedulers using web traffic. For these experiments, each site is downloaded over six concurrent connections using HTTP1 and page load time is used as metric. For the WLAN/3G setting, we observe that all schedulers perform similarly with marginal differences in most cases, with OTIAS providing a slight improvement in page load time compared to LRF for Wikipedia. For the WLAN/WLAN scenario, we observe increased page load times for DAPS and OTIAS for Huffpost, and also for ECF although marginal, compared to LRF.

Overall, we observe that the current state-of-the-art schedulers are limited to address the path asymmetry for different applications. For example, DAPS is not able to promptly react to network changes due to long schedule runs arising from high-delay asymmetry. Furthermore, it persistently uses all available subflows, even if their contribution is small. This is the main difference compared to both OTIAS and LRF, which can reduce or even skip the input of a certain subflow. On the other hand, OTIAS assumes symmetric forward delays, i.e. OWD = RTT/2 and makes decisions on a per-segment basis, using the current network state and the subflows' CWND as input. However, it builds up queues on subflows with the lowest RTTs, regardless of available CWND space. Moreover, similar to DAPS, OTIAS does not have a defined behaviour for retransmissions. ECF's design is very similar to OTIAS when determining whether packets should be scheduled in another subflow or wait, i.e. skipping a transmission opportunity. However, it has a better implementation design: It does not build queues on the subflows and it does not include simplifications such as OWD = RTT/2, which can be detrimental in real scenarios.

The rationale of this experiment is to illustrate that, despite being designed with asymmetric paths in mind, recently proposed schedulers do not achieve the performance that could have been expected. This justifies the need for new, more latency aware, schedulers. Furthermore, as neither OTIAS nor DAPS can significantly improve performance relative to LRF, and ECF brings clear benefits only with bulk in symmetric scenarios (see Figures 4 and 5); in the remainder of this paper, we take LRF as the baseline scheduler while evaluating the proposed schedulers.

IV. Low-Latency MPTCP Schedulers: Design and Implementation

By tackling path asymmetry to avoid out-of-order delivery, latency can be reduced significantly. However, current state-of-the-art schedulers are not able to completely do so. To mitigate latency problems, we introduce two new scheduling algorithms: BLEST [10], and STTF [11]. While BLEST is designed to estimate and minimise buffer blocking, STTF's goal it to predict and minimise the transfer time of each individual segment, to circumvent any blocking problems.

A. Blocking Estimation (BLEST)

The main idea behind BLEST is to keep the buffer blocking under control and avoid spurious retransmissions. In the following, we explain the design of BLEST's [10] new metric to estimate the amount of buffer blocking that might result from scheduling a segment on a given subflow. The BLEST algorithm is presented in Algorithm 1.

Algorithm 1 BLEST [10]

```
Require: $srtt_F < srtt_S$
1: if $can\_send(F)$ then
2:     $selected\_subflow = F$
3: else if $can\_send(S)$ then
4:     $rtts = srtt_S / srtt_F$
5:     $X = MSS_F \times (CWND_F + (rtts - 1)/2) \times rtts$
6:     if $X \times \lambda \leq MPTCP_{SW}\text{-}MSS_S \times (inflight_S + 1)$ then
7:         $selected\_subflow = S$
```

To overcome the reactive approach of MPTCP's default LRF scheduler, which reduces the CWND of slower subflows, we propose a proactive approach that decides at the time of segment scheduling whether to send over the slow subflow or not. The decision is made using MPTCP's send window.

MPTCP maintains a send window on its connection level for each connection, one level above the subflows. This window is necessary due to the full data multiplexing among all subflows belonging to the same MPTCP connection. However, due to the scheduler, if data is not acknowledged in one of the subflows, MPTCP's send window can be temporarily blocked, stalling the multi-path connection.

Thus, BLEST assumes that provided a segment is sent on S, the segment will occupy space in MPTCP's send window ($MPTCP_{\text{SW}}$) for at least RTT_S. We assume that all segments in flight on S occupy space in the window for the same amount of time. This is a conservative assumption since these segments can be acknowledged earlier on the fastest subflow. The remaining send window can be used by the faster subflow, i.e. the lower RTT subflow, F. This means that blocking would occur if F was not able to send, e.g. due to lack of space in the send window because of S. Therefore, we estimate the amount of data, X, that will be sent on F during RTT_S, and check whether this data fits into MPTCP's send window. To estimate X, we assume that for every RTT_F, its CWND grows by 1 (as it is done in congestion avoidance) and is always filled by the scheduler, as

$$\begin{aligned} rtts &= RTT_S / RTT_F \\ X &= MSS_F \cdot (CWND + (rtts - 1)/2) \cdot rtts \end{aligned}$$

If $X \times \lambda > |MPTCP_{\text{SW}}| - MSS_S \cdot (inflight_S + 1)$, the next segment will not be sent on S. Instead, the scheduler waits for the faster subflow to become available. Essentially, while LRF always opts to use an available subflow, BLEST is able to skip a subflow, waiting for a more advantageous subflow which can offer a lower risk of blocking, and the number of retransmissions that would have been consequently triggered.

The estimate of X can, however, sometimes be inaccurate. To address this, we introduce a correction factor λ, to scale X. The λ correction factor is adjusted as follows. Buffer blocking during one RTT_F is an event that triggers an increase of λ by δ_λ. Conversely, the absence of buffer blocking triggers a decrease by δ_λ. At the beginning of the connection, we set $\lambda = 1.0$, i.e. no correction of the estimation.

The implementation of BLEST along with all the necessary changes to MPTCP is available at [12]. The implementation consists of 800 lines of code and can be used with version 0.91.2 of the Linux MPTCP kernel.

B. Shortest Transfer Time First (STTF)

STTF's design is similar to OTIAS [21] and ECF [22] in that it tries to calculate the expected transfer time of a segment considering the path characteristics of all available subflows. Thus, STTF will schedule all unsent data on the fastest available subflow, even though its CWND is full. While the initial idea behind STTF is sketched in a technical report [11], this article presents the complete algorithm together with a Linux implementation and performance evaluation.

Conceptually, the idea behind STTF is simple: whenever there is data ready to schedule, STTF traverses all unscheduled segments; predicts the transfer time for each of them, when transferred over each of the available subflows, and determines, on the basis of the predicted shortest transfer time, on which subflow each segment should be scheduled. The predicted transfer time for a segment over a certain subflow is computed with regards to the SRTT, the congestion state of the subflow, i.e. whether the subflow is in slow-start or congestion avoidance, and the number of segments already queued in the subflow. The consideration of the current congestion state is an important difference to other schedulers that often assume that connections are in congestion avoidance, an assumption that is often wrong as short-lived flows, e.g. which often start and terminate before leaving slow-start.

After the assignment of segments to subflows, STTF releases its control to the MPTCP output engine to enable data transmission. If the transmission is interrupted in any way, e.g. by incoming acknowledgments before all segments have been transmitted, STTF removes all subflow assignments and reschedules unsent data. The rescheduling is done to update transmission time estimations as the external event(s) causing the interruption may have altered previous conditions. The rescheduling is a major component that enables STTF to react more promptly to changes in the underlying network than, e.g. DAPS, OTIAS, and ECF (see Section III-D). In addition to the congestion state optimisation and re-scheduling, STTF also makes use of some minor optimisations. These include, e.g. an RTT-reset mechanism to solve the problem of using stale RTT measurements, described in Section II-C.

Algorithm 2 STTF Scheduling

```
1: mptcp_sttf_reschedule()
2: for each unsent segment $p$ do
3:     for each available subflow $s$ do
4:         $T^p_s$ = transfer_time(s,p)
5:         if $T^p_s$ < $T_{min}$ then
6:             $T_{min}$ = $T^p_s$
7:             selected_subflow = s
```

The main loop of STTF is shown in Algorithm 2. First, it extracts all unsent data from all subflows for rescheduling.[1] Then, for each segment, it calculates the expected transfer time considering all subflows, scheduling the corresponding segment on the subflow with the shortest transmission time. The actual transmission time is calculated as Algorithm 3. The algorithm starts by calculating the time needed for transmission that can be immediately sent; in other words, segment transmission that is not limited by the CWND of

[1]In practice, the rescheduling of unsent data is not done in every scheduler invocation. For instance, if paths are symmetric, or a bulk flow is transferred, this operation is unnecessary and it is ignored to save computational resources.

the current subflow (lines 2–3). After this initial assessment, the algorithm calculates how much the CWND can grow during the current RTT, considering the subflow's congestion control state (line 5). The calculation considers if the subflow: (a) is, and will be, in slow start the entire RTT; (b) will exit slow start during the RTT; or (c) be completely in congestion avoidance. The remainder of the algorithm uses the predicted CWND together with the amount of data to send to derive a transmission time that is dependent on the number of RTTs the subflow will spend in different congestion states.

Algorithm 3 STTF Time Calculation

```
1: procedure TRANSFER_TIME
2:     if cwnd_free > 0 and data_to_send < cwnd_free then
3:         return rtt / 2
4:     transfer_time = transfer_time + rtt
5:     cwnd = increase_cwnd(current_cc_state)
6:     if data_to_send <= max_segments_in_ss then
7:         transfer_time = transfer_time + rtt × (rounds_in_ss-
               1) + rtt/2
8:         return transfer_time
9:     else
10:        if cwnd < ssthresh then
11:            transfer_time = transfer_time + max_rounds_in_ss
                   × rtt
12:            if ends_in_ss(data_to_send) then
13:                return transfer_time
14:            cwnd = ssthresh
15:        transmission_time += rtt × (rounds_in_ca - 1) + rtt
               / 2
16:    return transfer_time
```

The STTF implementation along with all the necessary changes to MPTCP is available in [13]. The implementation consists of about 1200+ lines of code and can be used with version 0.91.2 of the Linux MPTCP kernel.

C. Functional Comparison of LRF, BLEST, and STTF

We continue by elaborating on how scheduling decisions are made by LRF, BLEST, and STTF. The data shown in Figure 6 was collected from controlled emulation experiments using Linux with an MPTCP-enabled kernel equipped with all algorithms. Figure 6 illustrates how the schedulers act when a burst of 15 segments are sent by an application when two paths are available, and their respective RTTs are $P_1 = 10$ ms and $P_2 = 100$ ms. For the sake of simplicity, the available capacity is equal for both paths (0.5 Gbit/s). Furthermore, the two MPTCP subflows traversing these paths both have a CWND of ten segments. The graphs illustrate each invocation of the schedulers. That is, whenever the corresponding scheduler is utilised by MPTCP to schedule a segment, the graphs show for each path: (a) the smoothed RTT (SRTT) at the moment of scheduling (●); (b) if the segment cannot be scheduled due to full CWND (▼); or (c) if TSQ inhibits scheduling (★). The lines connecting the SRTT dots indicate both that a segment was successfully scheduled and over which path it will be sent.

First, we consider the LRF scheduling mechanism, as shown in Figure 6a. When scheduling the first six segments, LRF selects P_1 having a 10 ms SRTT, compared to the 100 ms SRTT over P_2. However, both the seventh and eight segments are inhibited from being sent over P_1, even though the SRTT of P_1 is lower than that of P_2. The reason is that TSQ has kicked in on the faster path, in an attempt to limit the queueing for this subflow. As a consequence, data is, instead, scheduled over the much slower subflow (P_2). After scheduling the two segments over the slower path, the queueing in lower layers has been cleared and the faster path is used again for four additional segments. After this point, however, the scheduler has filled the CWND of the faster subflow and is yet again forced to schedule data over the slower path, even though the transmission will be completed much slower.

The scheduling decisions by BLEST is made within two RTTs, compared to LRF. During the first RTT, shown in Figure 6b, BLEST only schedules data on P_1, as indicated by the black line connecting the ten scheduling invocations. When ten segments have been scheduled, and the CWND limitation kicks in, BLEST does not schedule data on P_2. Instead, BLEST waits for acknowledgments from the segments sent on P_1 to open up the CWND. When the acknowledgments arrive during the next RTT, shown in Figure 6c, BLEST again starts to schedule data on P_1. Notice that BLEST, during this RTT as well, rather waits for the CWND limit and TSQ to disappear instead of scheduling data over the slow subflow. This simple experiment illustrates BLEST's ability to prevent data from arriving out-of-order at the receiver.

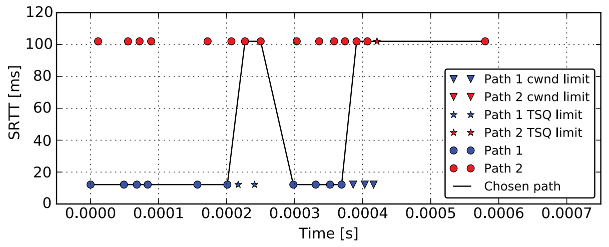
Finally, Figure 6d illustrates how STTF works. In contrast to the SRTT metric used by LRF and BLEST, STTF operates on its estimated transmission time (TT) of each individual segment. As described earlier, LRF is very sensitive to CWND limitations and chooses a slower subflow when the fastest filled its CWND. BLEST was shown not to pick a slower subflow, but rather wait for the faster one to become available again. In contrast to both LRF and BLEST, STTF does not care about CWND limitations or TSQ when scheduling. Instead, it relies on its rescheduling component to fix suboptimal decisions. Therefore, as shown in Figure 6, the scheduling occurs on the subflow with the lowest calculated transfer time, in this case always P_1, for all segments.

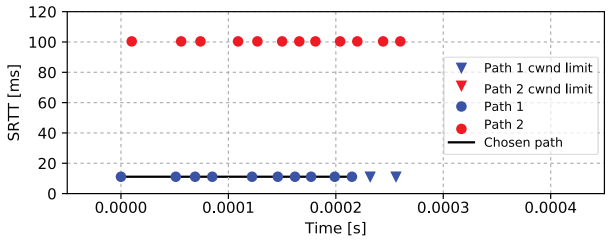
V. Evaluating BLEST and STTF

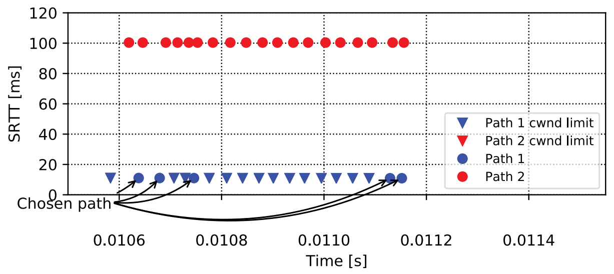
To assess the benefits of avoiding blocking and closely approximating the transmission time of queued data, we performed controlled experiments comparing the performance of LRF, BLEST, and STTF. To evaluate the schedulers regarding latency and throughput, we conducted a series of experiments involving synthetic and real workloads, including traffic from interactive applications and web browsing. The schedulers were also evaluated in real-world experiments and these results are reported in Section VI.

A. Experimental Setup

A testbed setup consisting of five regular desktop computers was used for the experiments. The machines were connected as illustrated in Figure 7, to emulate a topology with: a client, two wireless access points (WLAN/3G), a server access router and a server. Two configurations were used for the experiments; one for basic latency and throughput experiments (reported in Section V-B) and one for experiments involving web and interactive traffic (reported in Section V-C and V-D). The left part of Table II shows the parameterisation for the basic latency and throughput experiments. The parameters used for these experiments were not chosen to be representative


(a)


(b)


(c)

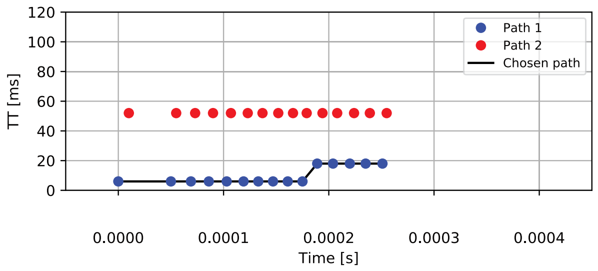

(d)

Fig. 6. Scheduling decisions for a burst of 15 segments, using the LRF, BLEST, and STTF schedulers. (a) LRF scheduler. (b) BLEST scheduler (1st RTT). (c) BLEST scheduler (2nd RTT). (d) STTF scheduler.

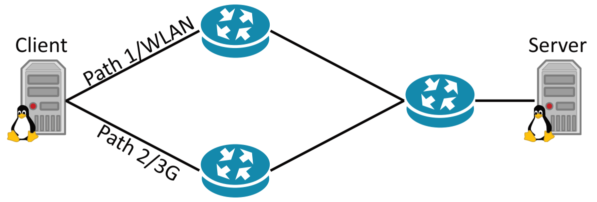


Fig. 7. Experimental multi-path setup.

TABLE II
Network Characteristics. For the Latency and Throughput Experiments (Left), Bandwidths and Delays Were Fixed During Each Experimental Run and for the Other Experiments (Right) the Characteristics Were Uniformly Distributed During Each Run

	Latency and Throughput (Section V-B)		Web and Google Maps (Sections V-C and V-D)	
	Path 1	Path 2	WLAN	3G
Capacity [Mbps]	100	100	20-30	3-5
Delay [ms]	10	10, ..., 400	5-25	65-75
Loss [%]	0	0	0-1	0

of typical WLAN/3G setups but rather to exhibit necessary path asymmetry, causing data to arrive out-of-order. The right part of Table II shows the parameterisation for the web and interactive traffic experiments. In contrast to the previous configuration, this one is meant to represent a typical WLAN/3G setup. The parameterisation has also been used in previous evaluation studies [17]. The WLAN links are 802.11ag, and the loss rate is the rate experienced on transport-level (not including, e.g. link-layer retransmissions).

To enable multi-path communication between client and server, the machines were equipped with MPTCP-enabled Linux kernels. The client used a stock version of the MPTCP kernel, and the server used a modified version including the BLEST and STTF schedulers. For all experiments, MPTCP was configured to use Path 1/WLAN as the primary path. The other machines in the setup used regular Linux kernels configured to emulate link characteristics and forward traffic. All network-related buffers were kept at their default settings.

B. Latency and Throughput

As a first step to assess the schedulers' latency performance, we compared the transmission time of differently sized bursts sent over MPTCP using two paths, varying their RTTs. We also ran the same experiments over TCP over the path with the smallest RTT. The bursts were sized between 10 and 50 segments and the path RTTs were varied in such a way that both symmetric and very asymmetric connections were evaluated (path 1 had a 10 ms RTT and path 2 had a 10, 40, and 100 ms RTT). Before each experiment, we pre-established the MPTCP connection and sent enough data to create and utilise subflows over both paths. This "warm-up" was performed to guarantee that two subflows, one per path, were available to MPTCP when experiments started. After each warm-up, the corresponding MPTCP connection was made idle long enough to reset the congestion state back to the slow-start phase. The experiments for each scheduler combination, burst size and path RTTs were repeated 30 times to minimise random variations. Figure 8 shows the results of these experiments. The y-axis of each graph shows the time (in ms) required for a burst to be transmitted from sender to receiver, and the x-axis shows the size of the corresponding burst (in segments). Note that the transmission times in the graph only show the one-way application delay, i.e. the time required for the data to travel from sender application to receiver application. The leftmost graph shows the results of the symmetric experiment where both paths had an RTT of 10 ms. The middle graph shows a slightly asymmetric scenario where path 1 had an RTT of 10 ms and path 2 an RTT of 40 ms. Finally, the rightmost graph contains the results from the experiments in a highly asymmetric setting where the RTT of path 1 is 10 ms and path 2 had an RTT of 100 ms.

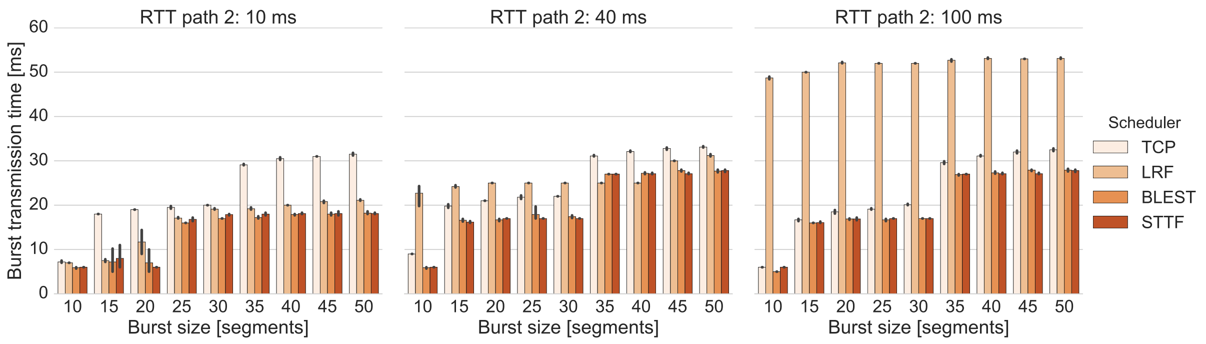


Fig. 8. Average transmission time for bursts of different sizes over subflows with symmetric and asymmetric RTTs.

MPTCP's default LRF scheduler is most suitable in settings with subflows that have symmetric characteristics. The reason why LRF performs almost on par with BLEST and STTF is simply due to the reason that symmetric connections are aligned to the strategy of LRF: divide the traffic equally among subflows. When the asymmetry increases (middle graph) LRF performance decreases, especially for bursts with 30 segments or less. One result that stands out is when 10 segments are transmitted, requiring significantly more time for LRF than for TCP, BLEST, and STTF. While this might seem strange at first, it is connected to the previously described use of TSQ to limit the queueing within the TCP stack, as shown in Figure 6. In this scenario, segments will be scheduled over the fast subflow until TSQ is temporarily active and inhibits the use of this subflow. As previously mentioned, in Section II-C, this happens whenever the queueing exceeds two segments or approximately one millisecond worth of data. When asymmetry further increases (rightmost graph), the results for LRF show quite high delays for all the different burst sizes, and it is evident that some data always is sent over the slower 100-ms RTT path. For most burst sizes this is due to CWND limitations. LRF, as previously explained, will immediately select another path if the CWND of the current one becomes filled. This fact is the case for nearly all depicted results, as the initial CWND in Linux is set to ten segments [23]. Thus, when there are more than ten segments to be sent, the use of the slow path is inevitable. This is not the case for BLEST and STTF, both of which opportunistically forego the use of the slow path. Interestingly, MPTCP also transmits data over the slowest when the amount of data is equal to the initial CWND (burst of ten segments).

Many of MPTCP's motivating use cases have focused on increasing throughput. Therefore, it is interesting to see whether BLEST and STTF can maintain good throughput while at the same time reducing latency. To test this, a simple bulk transfer application was used to transmit data between the sender and the client. Each transfer was configured to last for three minutes. Figure 9 shows the average goodput for TCP (over a single 10-ms RTT path) and MPTCP with the LRF, BLEST and STTF schedulers. The graph illustrates how the average goodput varied with increasing RTT on path 2. As seen in the experiments with symmetric or slightly asymmetric paths, MPTCP performed almost the same regardless of the scheduler. However, with increasing asymmetry, the BLEST goodput, and, even more so, the STTF goodput increased more than that of LRF. The problem for LRF is receiver buffer blocking: Segments arriving on the fast path is buffered while waiting for previously sent data on the slow path. In those cases the asymmetry is large and the flow long enough, the receive buffer eventually fills up and causes the sender to intermittently stop sending. Since BLEST is designed to reduce blocking effects, it is able to perform better than LRF for moderate asymmetry. The minimisation of the per-segment transmission time done by STTF is even more effective. As shown in the graph, STTF is almost able to match TCP, even when the asymmetry between the paths is large. The TCP results are included to show that path asymmetry, after a certain point, causes MPTCP goodput to be lower compared to single-path protocols.

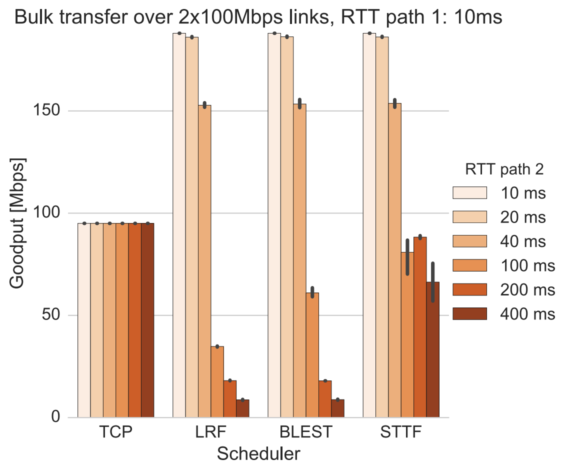


Fig. 9. Average goodput for TCP and MPTCP (LRF, BLEST, and STTF).

Although the STTF implementation is complex, we believe that the algorithm will work well, even for link rates at gigabit speeds. The calculations required by the STTF algorithm are straightforward and can be optimised if necessary. However, it is likely that the current implementation of STTF will underperform somewhat at gigabit speeds as the process of scheduling/rescheduling data causes segments to be moved back and forth between the MPTCP send queue and the subflows. The reason why the current implementation does this is one of convenience; there were already mechanisms available for this kind of operation within the Linux MPTCP implementation. However, to save computational resources the STTF implementation does not perform a rescheduling on every invocation of the scheduler. For instance, if paths are symmetric or bulk flows are transferred this operation is unnecessary and is therefore ignored. For a completely optimised version of STTF, ready for production use, the scheduling

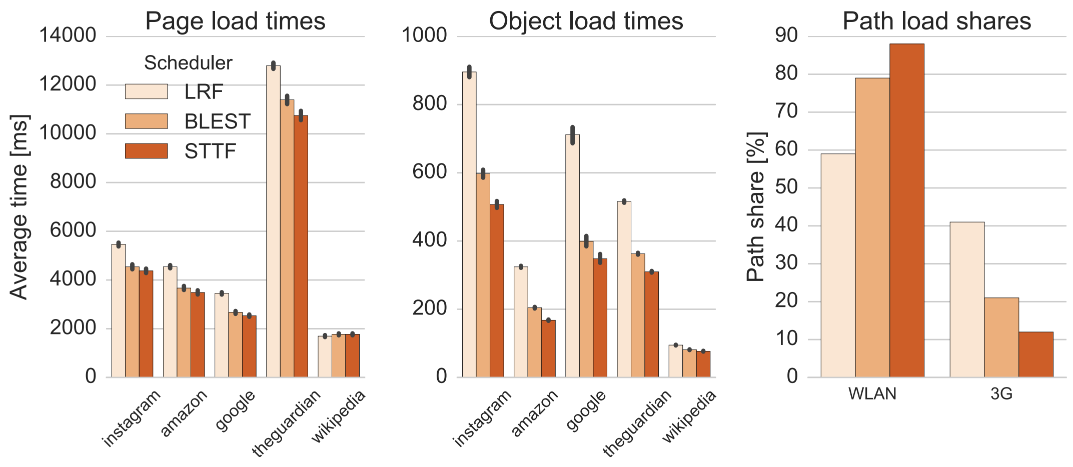


Fig. 10. Page load times, object download times and traffic path shares for HTTP2 experiments in emulated environment.

TABLE III
WEBSITES USED IN THE EVALUATION

Web site	Objects	Total size [Kbytes]
Google	6	906
Wikipedia	19	132
Instagram	30	1,671
Amazon	59	1,234
The Guardian	228	3,166

should not result in segments being moved at all. Instead, the scheduling process should merely result in a path assignment stored as a variable in the data structure of the corresponding segment. Segments should then be moved on demand when its corresponding subflow is ready for transmission. In the current implementation of STTF, rescheduling is not done on each invocation of the scheduler.

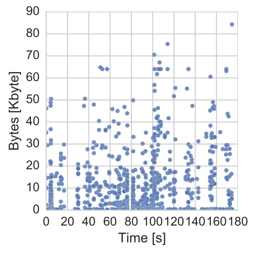
C. Web

Recently, HTTP2 [24] has gained momentum and seen a major increase in the number of supported domains. From its standardisation in May 2015 until today, the deployment has gone from approximately 13 000 to 242 000 hosts [25]. Compared with HTTP1, HTTP2 enables a more efficient use of network resources and can reduce latency by, e.g. providing header compression and concurrent object download over the same connection. The simultaneous download mechanism is interesting as it enables an MPTCP scheduler to simultaneously optimise the transmission over several paths. For the experiments in this section, we chose five websites of different sizes: google.com, wikipedia.com, instagram.com, amazon.com, and theguardian.com. The sites vary both regarding the number of objects and in their respective sizes. Details regarding the size distribution can be found in Table III. The primary goal of this experiment was to evaluate whether the proposed schedulers can reduce latency for web downloads, compared to LRF. To perform the actual experiments, we used a set of applications on the client and server to perform the downloads. On the server, we used the `nghttp2` web server which supports HTTP2 with, e.g. HPACK [26] header compression. To model the download process as thoroughly as possible we used data and dependency graphs from `Epload` [27] together with a custom-made client implemented with `libcurl`. The dependency models tell the client in which order to download the web objects and if there should be processing delays in between downloads. The models are based on how real browser implementations download and process these particular web sites. To account for variability, each experiment was repeated 30 times.

Figure 10 shows the average page load times, web object download times, and the traffic share over each path, all with 95% confidence intervals. The values are shown for the default LRF scheduler, BLEST, and STTF. Although a page becomes usable before it is completely loaded, the page load time metric is useful for comparing network performance. Furthermore, the page load times seem to correlate with the object download times (middle graph). We define the object download time as the average time required to download a single web object. The download time of an object is also a relevant metric, as users' browsing experience sometimes rely more on individual objects than entire pages. As shown in both graphs, LRF poses the worst performance of the three schedulers. BLEST decreases latency compared to LRF, and STTF further reduces latency for object downloads, with an exception to the Wikipedia site where the total amount of data was discovered to be too small to trigger different scheduling decisions among the schedulers, causing the object download times to be rather equal for all schedulers. For some sites, the performance improvement given by STTF is very significant. For instance, STTF completes web object transfers up to 51% faster than LRF for google.com. The reason for the differences in performance is that BLEST and STTF use the WLAN path more than LRF; the LRF scheduler sends approximately 60% of the data over the best path (WLAN), while BLEST uses this path for almost 80% and STTF for roughly 90% of the traffic.

D. Google Maps

Google Maps is an excellent example of an interactive application that requires low latency for a good user experience. To perform experiments with this application, we replayed real user sessions in our experimental environment. To obtain actual traffic, we used `tcpdump` and captured network traffic from repeated real sessions on a Google Chrome Web browser, where a person was visualising routes between arbitrary locations. Since the captured traffic largely depended


(a)

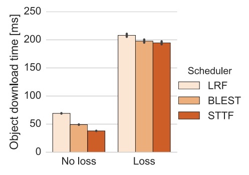

(b)

Fig. 11. Google Maps experiment, graphs showing both a sample of the traffic pattern and the results. (a) Google Maps data. (b) Average object download times.

on the network conditions, such as the available bandwidth and the RTT, at the time of the capture, we could not directly create traffic profiles from the captured data. Instead, we used the captures to reconstruct the application traffic and created traffic profiles from this traffic. Since the traffic was encrypted (HTTPS), the actual application traffic could only be approximated. The complete traffic generation process is further described in [28]. Figure 11a shows a representative traffic time series plot from the Google Maps experiment. Each dot in the graph represents a transmitted object, sent at x seconds and with a size of y bytes. As shown in the graph, the amount of data is small and sparsely communicated due to real user interactions. Figure 11b shows the average download time obtained with LRF, BLEST, and STTF, for objects included in the traffic workload. Similar to previous graphs, 95% confidence intervals are used, and the average download time is based on 30 repetitions. Since data loss had such a significant impact on performance, we choose to show the results from the experiments without data loss separately. For instance, when LRF is used and there is no data loss, the average download time is about 60 ms compared to more than 200 ms at 1% random packet loss. Similar numbers were not observed in any other experiment, and an analysis of the results confirms that the reason is the limited amount of data sent. When data loss occurs, and there is a small number of outstanding segments, MPTCP struggles to recover the lost data quickly [29]. Despite this, and that the differences due to the actual scheduling are smaller, there are still significant differences in performance. When no packet loss occurs, BLEST outperforms LRF by approximately 29%, and, when packets are lost, BLEST is about 5% faster. Similarly, when no packet loss occurs, STTF outperforms BLEST by approximately 23%, and, when packets are lost, STTF is about 1.5% faster. Compared to LRF, STTF is 45% faster when no packets are lost, and 6% faster otherwise.

VI. Real-World Experiments

This section evaluates LRF, BLEST, and STTF in real-world web experiments, using a mobile node with both WLAN and 3G interfaces. For these experiments, the MONROE mobile broadband (MBB) network measurement platform [30]–[32] was used. The MONROE platform provides a dedicated infrastructure for measuring and experimenting over MBB networks and WLAN. We used a setup similar to the one described in Section V-A, where the server was a remote web server located in Norway and the client was a MONROE node in Sweden. Both the server machine and the MONROE node were equipped with Linux MPTCP kernels, and the server additionally extended with BLEST and STTF. Before and during experimentation, we measured the RTT of the two paths, which showed an average of $RTT_{WLAN} \approx 25$ ms and $RTT_{3G} \approx 75$ ms. To compare the results to the previously reported (emulation) results we used the same software toolchain and websites as for these experiments (see Section V-C).

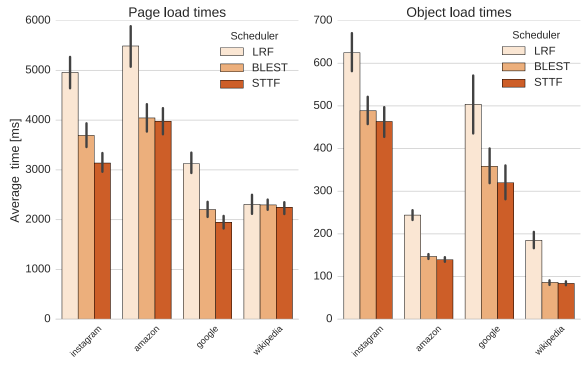


Fig. 12. Page load times and object download times for real-world HTTP2 experiments.

The results of the experiments are shown in Figure 12. The graphs show both the average page load times (left) and the average object download times (right) for 30 repetitions of downloading each site using the different schedulers. Similar to the emulation results, BLEST and STTF outperform LRF. For example, both BLEST and STTF reduce the page load times for Amazon with approximately 27%, and the object load times with an average of 41%. STTF achieves the best overall performance, although the variation in the results is larger when running over real networks.

For the Wikipedia experiments, there are no noticeable differences in page load times among the schedulers, although BLEST and STTF significantly reduce the object download times. This is interesting for two reasons. First, in the emulated web experiments the total amount of data to transfer was discovered to be too small to trigger different scheduling decisions among the schedulers, causing the object download times to be rather equal for all schedulers. Second, reduced object download times does not seem to cause a similar reduction in page load times. The reason for BLEST and STTF to transfer objects more quickly is related to the dynamics of the real networks used in these experiments. Variations in the real networks are more pronounced than in the emulated scenario, e.g. causing the RTTs of the different paths to fluctuate more. When examining packet traces from the experiments, it became evident that network fluctuations caused the data transmission to be more bursty, resulting in scheduling decisions for relatively larger amounts of data, at each instance, than in the emulated scenario. The larger amounts of data then caused different scheduling decisions, favourable to BLEST and STTF. The reason why the reduced object download times for BLEST and STTF did not translate into corresponding reductions in page load times is related to the Wikipedia site structure. As described in Section V-C, a dependency model is used for downloading each site. For the Wikipedia experiments, we could observe that a slightly quicker transmission of the objects was overshadowed by the dependencies and processing delays provided by the model.

The distribution of data over WLAN and 3G is similar to the emulation results, as shown in Figure 13. The difference among the schedulers is slightly lower than for the emulation

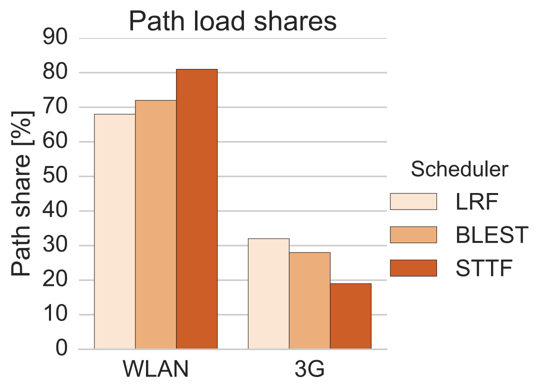


Fig. 13. Path load share between WLAN and 3G for real-world HTTP2 experiments.

experiments, with LRF scheduling approximately 68% of the traffic over WLAN, BLEST a little over 70%, and STTF slightly over 80%. The narrow span of load shares, as compared to the emulation experiments, is due to the variations in the underlying networks, sometimes causing the 3G path to becoming the preferred scheduling choice.

VII. Related Work

The authors in [33] present a survey identifying the most important aspects when shifting from single to multi-path transport. They offer a list of merits of multi-path transport, including features such as load balancing and diversification. To evaluate the expected benefits of such features, various studies have evaluated the performance of MPTCP in the wild [9], [34]–[36]. These studies found that MPTCP can provide higher throughput than TCP.

Even if the increased capacity through resource pooling with MPTCP could be measured, the relevance of MPTCP for latency-sensitive traffic is not straightforward. There are many sources of latency [37], some that might have a detrimental effect on the performance of transport protocols. For instance, HoL-blocking due to asymmetric paths can result in a connection experiencing an aggregate performance worse than that of its worst link, both regarding throughput and latency.

In relation to latency, the authors in [36] conclude that delay-sensitive applications using MPTCP's default scheduler need fairly symmetric paths in terms of bandwidth and packet loss, to gain from a path with shorter RTT. A similar finding is made in [9], which shows that realistic scenarios with asymmetric links do not make MPTCP perform better than a single-path protocol. Both [34] and [35] show that paths with shorter RTTs are used more frequently by MPTCP, resulting in lower latency, if the CWND is not a limiting factor. The latency reduction for cloud-based mobile applications has been assessed in [28], and for the streaming of high-quality mobile video in [38]. The authors in [28] find that MPTCP can reduce latency for delay-sensitive applications provided that paths are symmetric. Since asymmetric paths hamper performance, they also identify the need for a different scheduler. To overcome issues from path asymmetry, Wu *et al.* [38] argue for significant modifications to MPTCP's retransmission mechanism.

Yedugundla *et al.* [17] evaluate the adequacy of using both CMT-SCTP and MPTCP as transport for video, web, and gaming traffic, and concludes that, in most cases, the delay is not increased by using multiple paths. This article demonstrates that improving MPTCP scheduling is a crucial step to reduce multi-path transport latency. This is also confirmed by [39], which observes that the best scheduling policy depends on the underlying path characteristics. Indeed, the occurrence of out-of-order, and thus blocking, can increase when the asymmetry increases. It is worth pointing out that depending on the cellular technology, the asymmetry between WLAN and cellular can change dramatically, even without considering the *bufferbloat* effect on the cellular links [40], [41].

The most commonly used MPTCP schedulers are compared in [18], where asymmetry is identified as one of the causes of poor scheduling performance. To tackle this problem, researchers have taken different approaches. For instance, efforts have been made to optimise scheduling for specific types of traffic. For video application traffic, several studies have been conducted [42]–[46]. The performance that can be achieved with knowledge of the transmitted data, such as shown in [44], suggests that providing more proactive, cross-layer information between applications and the transport layer is necessary, and could be extended to other types of applications as well. The exchange of cross-layer information is an interesting approach, especially in managed environments such as data centers [47]. The more traditional approach is to reduce HoL-blocking without any application-specific knowledge [18], [21], [22], [48]–[50]. The most relevant schedulers in relation to our work have been detailed earlier in this work, including LRF [18], DAPS [49], OTIAS [21], and ECF [22]. While DAPS was an initial attempt to reduce HoL-blocking over asymmetric links, it was later shown to generate a non-negligible amount of spurious retransmissions. To cope with this, BLEST estimates the amount of receiver buffer blocking for a given scheduling of segments on the available paths. OTIAS was an initial attempt to minimise transmission times by estimating the time required to send individual segments over different subflows. STTF furthers the work on OTIAS by adding a more precise time estimation and allows rescheduling of segments to better accommodate for changes in the network conditions. ECF is a recent scheduler that tries to maximise throughput by increasing the use of the fastest subflow. This is done by sometimes skipping transmission over slower paths, and instead wait for the fastest path to become available. ECF is very similar to OTIAS and does not consider slow-start or rescheduling.

VIII. Conclusions

Today's networks are often multi-path, i.e. an end host can reach its peer through more than one network path. For example, mobile devices usually have multiple interfaces and data centers have redundant paths between servers. In view of this, several works have investigated how to exploit this increased connectedness to aggregate capacity and improve robustness. While capacity aggregation works well for network paths with symmetric capacity and delays, it has been shown to limit the throughput and increase delays when using asymmetric link technologies (e.g. LTE and WLAN). The reason for poor performance can often be found in the scheduler that decides over interface to transmit data. Basically, schedulers tend to overuse slower network paths causing data travelling over the faster paths to be blocked. This article provides an in-depth evaluation of three state-of-the-art schedulers (DAPS, OTIAS, and ECF) for MPTCP to highlight the problems of asymmetry. To alleviate asymmetry-related problems two novel schedulers, BLEST and STTF, are presented. BLEST estimates and tries to reduce buffer blocking effects, while STTF predicts and minimises the transmission time of individual segments. Both schedulers can provide significant latency reductions to interactive applications, such as web browsers, while still being able to provide throughput comparable to that of throughput-aware schedulers. Future work includes refinements to BLEST and STTF to better respond to varying link conditions.

REFERENCES

[1] "Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021," Cisco Syst., San Jose, CA, USA, White Paper, Feb. 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html

[2] B. Jin *et al.*, "Aggregating LTE and Wi-Fi: ToWARD iNTRA-cELL fAIRNESS AND hIgh TCP performance," *IEEE Trans. Wireless Commun.*, vol. 16, no. 10, pp. 6295–6308, Oct. 2017.

[3] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, *TCP Extensions for Multipath Operation With Multiple Addresses*, document RFC 6824, Jan. 2013.

[4] O. Bonaventure and S. Seo, "Multipath TCP deployments," *IETF J.*, vol. 12, pp. 24–27, Nov. 2016.

[5] Apple, "Advances in networking," in *Proc. Worldwide Developers Conf. (WWDC)*, Jun. 2017. [Online]. Available: https://developer.apple.com/videos/wwdc2017/

[6] C. Paasch and O. Bonaventure, "Multipath TCP," *Commun. ACM*, vol. 57, no. 4, pp. 51–57, Apr. 2014.

[7] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. 8th USENIX Conf. Networked Syst. Design Implement. (NSDI)*, Boston, MA, USA, pp. 99–112, 2011.

[8] C. Raiciu *et al.*, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM Conf.*, Toronto, ON, Canada, Aug. 2011, pp. 266–277.

[9] S. Ferlin, T. Dreibholz, and Ö. Alay, "Multi-path transport over heterogeneous wireless networks: Does it really pay off?" in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Austin, TX, USA, Dec. 2014, pp. 4807–4813.

[10] S. Ferlin, Ö. Alay, O. Mehani, and R. Boreli, "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks," in *Proc. FIP Netw. Conf. Workshops*, Vienna, Austria, May 2016, pp. 431–439.

[11] D. Hayes *et-al.*, "Report on prototype development and evaluation of end-system, application layer and API mechanisms," Simula Res. Lab., Oslo, Norway, Tech. Rep. RITE EU FP7-ICT, Sep. 2015. [Online]. Available: https://riteproject.files.wordpress.com/2015/12/rite_deliverable_1-3.pdf

[12] S. Ferlin and C. Paasch. (2017). *MPTCP BLEST Linux Kernel Implementation*. [Online]. Available: https://git.cs.kau.se/pub/sttf

[13] P. Hurtig and K.-J. Grinnemo. (2017). *MPTCP STTF Linux Kernel Implementation*. [Online]. Available: https://git.cs.kau.se/pub/sttf

[14] C. Raiciu, D. Wischik, and M. Handley, "Practical congestion control for multipath transport protocols," Dept. Comput. Sci., Univ. College London, London, U.K., Tech. Rep., Nov. 2009.

[15] C. Paasch *et al.*, (2016). *Multipath TCP in the Linux Kernel*. [Online]. Available: http://www.multipath-tcp.org

[16] O. Bonaventure, C. Paasch, and G. Detal, "*Use Cases and Operational Experience With Multipath TCP*," document RFC 8041, Jan. 2017.

[17] K. Yedugundla *et al.*, "Is multi-path transport suitable for latency sensitive traffic?" *Comput. Netw.*, vol. 105, pp. 1–21, Aug. 2016.

[18] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath TCP schedulers," in *Proc. ACM SIGCOMM Capacity Sharing Workshop (CSWS)*, 2014, pp. 27–32.

[19] E. Dumazet. (2012). *TCP Small Queues*. [Online]. Available: https://lwn.net/Articles/507065/

[20] G. Sarwar, R. Boreli, E. Lochin, A. Mifdaoui, and G. Smith, "Mitigating receiver's buffer blocking by delay aware packet scheduling in multipath data transfer," in *Proc. Int. Conf. Adv. Inf. Netw. Appl. Workshops (WAINA)*, Barcelona, Spain, Mar. 2013, pp. 1119–1124.

[21] F. Yang, Q. Wang, and P. D. Amer, "Out-of-order transmission for in-order arrival scheduling for multipath TCP," in *Proc. Int. Conf. Adv. Inf. Netw. Appl. Workshop (WAINA)*, Victoria, BC, Canada, May 2014, pp. 749–752.

[22] Y.-S. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "ECF: An MPTCP path scheduler to manage heterogeneous paths," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Modeling Comput. Syst.*, Urbana-Champaign, IL, USA, Jun. 2017, pp. 33–34.

[23] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, *Increasing TCP's Initial Window*, document RFC 6928, Apr. 2013.

[24] M. Belshe, R. Peon, and M. Thomson, *Hypertext Transfer Protocol Version 2 (HTTP/2)*, document RFC 7540, May 2015.

[25] M. Varvello *et al.*, "Is the Web HTTP/2 yet?" in *Proc. Int. Conf. Passive Act. Netw. Meas. (PAM)*, Crete, Greece, Apr. 2016, pp. 218–232.

[26] R. Peon and H. Ruellan, *HPACK: Header Compression for HTTP/2*, document RFC 7541, May 2015.

[27] X. S. Wang, A. Balasubramanian, A. Krishnamurthy, and D. Wetherall. (2014). *How Speedy is SPDY?* [Online]. Available: http://wprof.cs.washington.edu/spdy

[28] K.-J. Grinnemo and A. Brunstrom, "A first study on using MPTCP to reduce latency for cloud based mobile applications," in *Proc. 6th IEEE Int. Workshop Perform. Eval. Commun. Distrib. Syst. Web Based Service Architectures (PEDISWESA)*, Jul. 2015, pp. 64–69.

[29] M. Rajiullah, P. Hurtig, A. Brunstrom, A. Petlund, and M. Welzl, "An evaluation of tail loss recovery mechanisms for TCP," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 1, pp. 5–11, 2015.

[30] (2017). *EU H2020 MONROE Measuring Mobile Broadband Networks in Europe*. [Online]. Available: http://www.monroe-project.eu

[31] Ö. Alay *et al.*, "Measuring and assessing mobile broadband networks with MONROE," in *Proc. Int. Symp. A World Wireless, Mobile Multimedia Netw. (WoWMoM)*, Coimbra, Portugal, Jun. 2016, pp. 1–3.

[32] Ö. Alay, "Experience: An open platform for experimentation with commercial mobile broadband networks," in *Proc. ACM MobiCom*, Snowbird, UT, USA, Oct. 2017, pp. 70–78.

[33] S. Habib *et al.*, "The past, present, and future of transport-layer multipath," *CoRR*, vol. abs/1601.06043, Jan. 2016. [Online]. Available: https://arxiv.org/abs/1601.06043

[34] Y.-C. Chen *et al.*, "A measurement-based study of multipath TCP performance over wireless networks," in *Proc. ACM Internet Meas. Conf. (IMC)*, Barcelona, Spain, Oct. 2013, pp. 455–468.

[35] Q. De Coninck, M. Baerts, B. Hesmans, and O. Bonaventure, "Observing real smartphone applications over multipath TCP," *IEEE Commun. Mag.*, vol. 54, no. 3, pp. 88–93, Mar. 2016.

[36] B. Han, F. Qian, S. Hao, and L. Ji, "An anatomy of mobile Web performance over multipath TCP," in *Proc. ACM CoNEXT*, Heidelberg, Germany, Dec. 2015, Art. no. 5.

[37] B. Briscoe *et al.*, "Reducing Internet latency: A survey of techniques and their merits," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 3, pp. 2149–2196, 3rd Quart. 2016.

[38] J. Wu, C. Yuen, B. Cheng, M. Wang, and J. Chen, "Streaming high-quality mobile video with multipath TCP in heterogeneous wireless networks," *IEEE Trans. Mobile Comput.*, vol. 15, no. 9, pp. 2345–2361, Sep. 2016.

[39] B. Arzani, A. Gurney, S. Cheng, R. Guerin, and B. T. Loo, "Impact of path characteristics and scheduling policies on MPTCP performance," in *Proc. IEEE AINA Workshop*, May 2014, pp. 743–748.

[40] J. Garcia, S. Alfredsson, and A. Brunstrom, "Delay metrics and delay characteristics: A study of four Swedish HSDPA+ and LTE networks," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Paris, France, Jun./Jul. 2015, pp. 234–238.

[41] J. Huang *et al.*, "An in-depth study of LTE: Effect of network protocol and application behavior on performance," in *Proc. ACM SIGCOMM*, Hong Kong, 2013, pp. 363–374.

[42] T. Ojanperä and J. Vehkaperä, "Network-assisted multipath DASH using the distributed decision engine," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Feb. 2016, pp. 1–6.

[43] P. Houzé, E. Mory, G. Texier, and G. Simon, "Applicative-layer multipath for low-latency adaptive live streaming," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7.

[44] X. Corbillon, R. Aparicio-Pardo, N. Kuhn, G. Texier, and G. Simon, "Cross-layer scheduler for video streaming over MPTCP," in *Proc. 7th Int. Conf. Multimedia Syst. (MMSys)*, Klagenfurt, Austria, May 2016, Art. no. 7.

[45] C. Diop and G. Dugué, C. Chassot, and E. Exposito, "Qos-oriented MPTCP extensions for multimedia multi-homed systems," in *Proc. IEEE AINA Workshop*, May 2012, pp. 1119–1124.

[46] D. Jurca and P. Frossard, "Video packet selection and scheduling for multipath streaming," *IEEE Trans. Multimedia*, vol. 9, no. 3, pp. 629–641, Apr. 2007.

[47] M. Khabbaz, K. Shaban, and C. Assi, "Delay-aware flow scheduling in low latency enterprise datacenter networks: Modeling and performance analysis," *IEEE Trans. Commun.*, vol. 65, no. 5, pp. 2078–2090, May 2017.

[48] A. Alheid, A. Doufexi, and D. Kaleshi, "A study on MPTCP for tolerating packet reordering and path heterogeneity in wireless networks," in *Proc. Wireless Days (WD)*, Mar. 2016, pp. 1–7.

[49] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, and O. Mehani, "DAPS: Intelligent delay-aware packet scheduling for multipath transport," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Sydney, NSW, Australia, Jun. 2014, pp. 1222–1227.

[50] U. Javed, M. Suchara, J. He, and J. Rexford, "Multipath protocol for delay-sensitive traffic," in *Proc. 1st Int. Commun. Syst. Netw. Workshops*, Jan. 2009, pp. 1–8.

Per Hurtig received the M.Sc. and Ph.D. degrees in computer science from Karlstad University, Sweden, in 2006 and 2012, respectively. He is currently an Associate Professor at the Department of Computer Science, Karlstad University. His research interests include transport protocols, low-latency Internet communication, multi-path transport, and network emulation. He has participated in several international research projects and is also involved in Internet standardization within the IETF.

Karl-Johan Grinnemo (SM'11) received the Ph.D. degree in computer science from Karlstad University in 2006. He has worked almost 15 years as an Engineer in the telecom industry; first at Ericsson and then as a Consultant at Tieto. A large part of his work has been related to Ericsson's signaling system in the mobile core and radio access network. Since 2014, he has been a Senior Lecturer at Karlstad University. His research primarily targets application- and transport-level service quality. In recent years, his research has to a large degree focused on the use of multipath transport protocols such as multipath TCP to increase reliability and throughput and decrease latency in IP networks. He has authored and co-authored around 40 conference and journal papers, and is a Senior Member of IEEE.

Anna Brunstrom received the B.Sc. degree in computer science and mathematics from Pepperdine University, Malibu, CA, USA, in 1991, and the M.Sc. and Ph.D. degrees in computer science from the College of William & Mary, Williamsburg, VA, USA, in 1993 and 1996, respectively. She joined the Department of Computer Science, Karlstad University, Sweden, in 1996, where she is currently a Full Professor and the Research Manager of the Distributed Systems and Communications Research Group. She has a background in distributed systems, but her main area of work over the last years has been in computer networking with a focus on transport protocol design, QoS issues, cross-layer interactions, wireless communication, and network security. She has authored/co-authored 10 book chapters and over 100 international journal and conference papers.

Simone Ferlin received the Dipl.-Ing. degree in information technology with major in telecommunications from Friedrich-Alexander Erlangen-Nuernberg University, Germany, in 2010, and the Ph.D. degree from the University of Oslo, Norway, in 2017. She is currently a Researcher at the Network Architecture and Protocols Group, Ericsson Research. Her research interests lie in the areas of computer networks, transport protocols, congestion control, network performance, security, and measurements. Her dissertation focused on improving robustness in multipath transport for heterogeneous networks with MPTCP.

Özgü Alay received the B.S. and M.S. degrees in electrical and electronic engineering from Middle East Technical University, Turkey, and the Ph.D. degree in electrical and computer engineering from the Tandon School of Engineering, New York University. She is currently a Senior Research Scientist at Networks Department, Simula Research Laboratory, Norway, and an Associate Professor with the University of Oslo, Norway. She has authored more than 50 peer-reviewed IEEE and ACM publications and she actively serves on technical boards of major conferences and journals. Her research interests lie in the areas of mobile broadband networks, multi-path protocols and robust multimedia transmission over wireless networks.

Nicolas Kuhn received the master's degree in aeronautical engineering in 2010. From 2010 to 2013, he was a Ph.D. student at the Institut Supérieur de l'Aéronautique et de l'Espace (ISAE) and at the National ICT Australia (NICTA) to obtain a Ph.D. from the University of Toulouse - EDMITT in December 2013. From 2014 to 2015, he was the Principal Investigator with the Institut Mines-Télécom, Télécom Bretagne for the RITE European Project. Since 2015, he has been with the Centre National d'Etudes Spatiales as a Research Engineer. His research includes transport layer issues in spatial telecommunications and how the end-to-end service can be achieved in this challenging environment. Thus, he also works on quality of experience, quality of service, access methods, and cross-layer designs.