BLEST: Blocking Estimation-based MPTCP Scheduler for Heterogeneous Networks

Simone Ferlin,*†, Özgü Alay*
*Simula Research Laboratory, Norway
{ferlin,ozgu}@simula.no

Olivier Mehani,† Roksana Boreli†
†National ICT Australia (NICTA), Sydney, Australia
{first.last}@nicta.com.au

***Abstract*—With the widespread availability of multi-homed devices, multipath transport protocols such as MPTCP are becoming increasingly relevant to support better use of multiple connectivity through capacity aggregation and seamless failover. However, capacity aggregation over heterogeneous paths, such as offered by cellular and Wi-Fi networks, is problematic. It causes packet reordering leading to head-of-line (HoL) blocking at the receiver, increased end-to-end delays and lower application goodput. MPTCP tackles this issue by penalising the use of longer paths, and increasing buffer sizes. This, however, results in suboptimal resource usage. In this paper, we first evaluate and compare the performance of default MPTCP and alternative state-of-the-art schedulers, all implemented in the Linux kernel, for a range of traffic patterns and network environments. This allows us to identify shortcomings of various approaches. We then propose a send-window *BL*ocking *EST*imation scheduler, *BLEST*, which aims to minimise HoL-blocking in heterogeneous networks, thereby increasing the potential for capacity aggregation by reducing the number of spurious retransmissions. The resulting scheduler allows an increase by 12% in application goodput with bulk traffic while reducing unnecessary retransmissions by 80% as compared to default MPTCP and other schedulers.**

***Index Terms*—MPTCP, multipath, transport protocol, packet scheduling, head-of-line blocking, receive window limitation, heterogeneous networks**

I. Introduction

Multipath transport protocols, and particularly Multipath TCP, allow to better use the network resources available to multi-homed devices such as mobile phones. Two main advantages are envisioned: capacity aggregation across multiple links, and the ability to maintain connection if one of the path fails. Capacity aggregation is however challenging with heterogeneous paths, such as offered by cellular and Wi-Fi, in particular because of delay heterogeneity [1]. This heterogeneity results in packet reordering, leading to head-of-line (HoL) blocking, increased out-of-order (OFO) buffer use at the receiver and, ultimately, reduced goodput.

MPTCP's default scheduler, *minRTT*, is based on Round-Trip Time (RTT). minRTT starts by filling the congestion window (CWND) of the subflow with the lowest RTT before advancing to other subflows with higher RTTs. When one of these subflows blocks the connection, *e.g.*, due to head-of-line blocking, MPTCP's default scheduler retransmits the segments blocking the connection on the lowest-delay path and penalise longer (*i.e.*, higher-delay) paths that caused the issue [2]. This has a long-term impact on the CWND of these subflows, which are limited in their growth [3], leading to sub-optimal capacity aggregation, as higher-delay paths are underused [4]. As a rule-of-thumb, it is also recommended to increase the receive buffer size to further limit HoL-blocking situations [5].

The need for multipath transport protocol schedulers is known, and a number of proposals have been made and evaluated in the past [6]. However, in the specific case of heterogeneous paths, more care is required to avoid the issues discussed above. Such schedulers have been proposed in [7]–[9], based on the concept of sending packets out of order so they reach the receiver in order. There exists, however, no comparison of these schedulers to the MPTCP default scheduler in a consistent environment.

In this paper, we first offer a comparative study of the proposed MPTCP schedulers [7]–[9], by experimentally evaluating our Linux implementation of these algorithms. We evaluate their behaviour for different traffic types (Web, Bulk, CBR). The performance of these schedulers is compared to MPTCP's default scheduler as well as plain single-path TCP, in terms of application goodput (for bulk traffic), end-to-end delays (CBR) and completion time (Web). Based on observations in these experiments, we identify how the studied mechanisms offer the best performance, and what they fail to properly account for. We also take insight from the observations of [10] that not all subflows should be used at all times and, while scheduling is needed to complement pure congestion control, path selection and send buffer management are also primordial. We then propose a novel *BL*ocking *EST*imation-based scheduler, *BLEST*, which takes a proactive stand towards minimising HoL-blocking. Rather than penalising the slow subflows, BLEST estimates whether a path will cause HoL-blocking and dynamically adapts scheduling to prevent blocking. Although BLEST is designed for heterogeneous paths, we show in our experiments that it works as well as MPTCP's minRTT scheduler in homogeneous scenarios.[1]

The remainder of this paper is organised as follows. We present the background to this work, and show motivating examples in the next section. We describe our evaluation setup in Section III. In Section IV, we discuss our implementation of different schedulers [7]–[9] and compare their performance side-by-side with MPTCP's default scheduler. Based

ISBN 978-3-901882-83-8 © 2016 IFIP

[1]BLEST's code is available at http://nicta.info/mptcp-blest.

on observations in these experiments, we propose a proactive minimum-delay scheduler that can predict the send-window blocking risk, and schedule accordingly in Section V, and evaluate its performance in Section VI, both in emulated and real multipath environments. We finally offer some concluding remarks in Section VII.

II. Background and Motivation

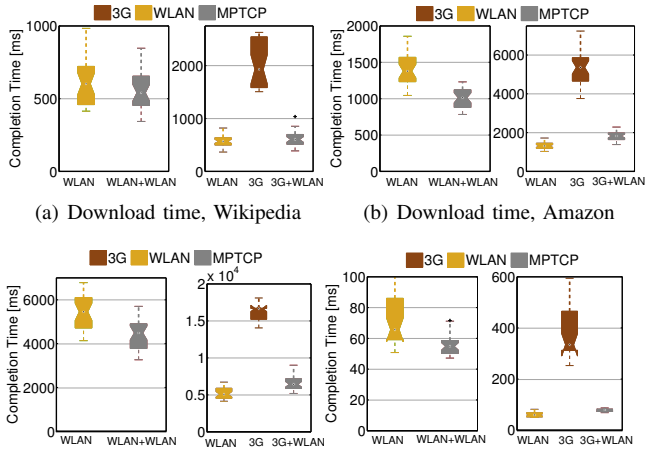
A. Multipath Transfer over Heterogeneous Paths

Multipath transport has be shown to provide benefits from bandwidth aggregation to increased robustness [2], [11]–[13]. Whenever the underlying network paths are homogeneous, MPTCP accomplishes its goals [14]. However, path heterogeneity can hinder achievement of MPTCP's goals, mostly due to the HoL-blocking which causes higher end-host memory usage and path bandwidth underutilisation [1], [3]. In MPTCP, the *scheduler* is the component that is responsible for the distribution of packets among the available paths. A well-designed scheduler that can dynamically adapt packet distribution based on the channel conditions to provide a better performance, both in terms of goodput and delay, is crucial.

MPTCP's default minRTT scheduler[2] first sends data on the subflow with the lowest RTT estimation, until it has filled its congestion window [2]. Data is sent on the subflow with the next higher RTT. In order to address the heterogeneity of the paths, a mechanism of *opportunistic retransmission and penalisation* (PR) has also been proposed in [2]. In order to quickly overcome HoL-blocking, opportunistic retransmission immediately reinjects segments causing HoL-blocking onto a subflow with an RTT lower than that of the blocking subflow which has space available in its congestion window. The penalisation mechanism also halves the congestion window of the blocking subflow to limit its use. [3] showed that MPTCP's PR does not behave well in some scenarios when path characteristics (*e.g.*, capacity, delay and loss rates) are significantly different. Penalisation of a long subflow (higher RTT) has a long-term detrimental impact on the performance: it will take longer for the subflow to increase its CWND, leading to underutilisation of the path and, ultimately, lower capacity aggregation.

In order to illustrate the challenges in heterogeneous scenarios, we ran experiments with constant bitrate (CBR) and web transfers, and contrast the results with homogeneous scenarios. In Figure 1, we observe that the amount of data and the path heterogeneity are the main factors determining the performance of MPTCP. MPTCP generally provides lower completion times, especially for websites with many objects. However, when the paths are heterogeneous in terms of delay and loss, as in the 3G+WLAN case, losses in the WLAN force MPTCP to use the 3G path, therefore MPTCP's completion time becomes higher than TCP on the WLAN path. Similarly, Figure 1(d) shows the same effect for the packet delay of a CBR flow: MPTCP's minRTT adequately leverages the aggregation of two homogeneous WLAN paths and reduces

[2]We base our work on MPTCP v0.90 throughout this paper.



(a) Download time, Wikipedia (b) Download time, Amazon

(c) Download time, Huffington Post (d) Packet delay, CBR video

Figure 1. Download times for selected websites, and application packet delay for CBR video traffic, both over MPTCP in **WLAN+WLAN** (left of each pair) and **3G+WLAN** (right of each pair) (CORE emulation, with background traffic, see III-1). MPTCP with heterogeneous paths (**3G+WLAN**) underperforms single-path TCP on the best (WLAN) path.

both delay and jitter; however, it doesn't perform as well as a single WLAN path when running over heterogeneous 3G+WLAN paths.

This goes against one of MPTCP's design goals: "[a] multipath flow should perform at least as well as a single path flow would on the best of the paths available to it" [5].

B. Schedulers for heterogeneous paths

Alternative multipath scheduling algorithms have been object of multiple studies [4], [10], [15]. In [6], the authors evaluated different scheduling strategies (pull, push and hybrid) focusing on implementation performance. They also considered how schedulers should cope with paths that have heterogeneous delay and/or capacities. They concluded that a scheduler must take both delay and capacity into consideration in order to effectively leverage multipath scenarios.

Later, [8] evaluated and extended the idea of a Delay-Aware Packet Scheduler (DAPS) [7] for MPTCP in order to overcome HoL-blocking due to path heterogeneity. In that work, the authors derived a rule-of-thumb for buffer size for MPTCP. [9] explored a more ambitious scheduler implementation, sending packets out of order so they arrive in order. They however included some simplications that expose vulnerabilities of the approach. For example, no consideration is given to segment reinjection if a certain path is blocking the connection.

These alternative algorithms were so far not extensively tested against MPTCP's default scheduler. The number of scenarios in which they were evaluated was also limited, and did not cover many scenarios (homogeneous vs. heterogeneous) and traffic classes. The differences in evaluation methods also make it difficult to accurately compare their performance. In the next sections, we address this issue by re-implementing these schedulers in the Linux kernel, and systematically evaluating their performance in a range of scenarios and traffic use-cases agains MPTCP's default scheduler.

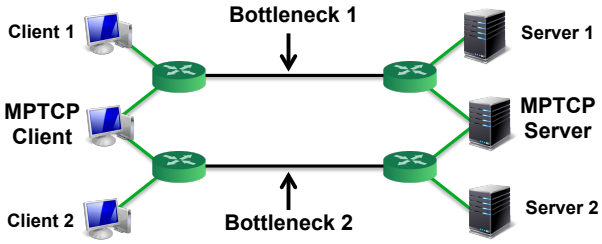


Figure 2. Emulation experiment setup

III. Measurement Setup

We used CORE [16] for the initial evaluation. CORE is a network emulator able to emulate a real network stack implementation within Linux containers, making it suitable to avoid simulation model simplifications. Figure 2 shows the emulation topology. Bottleneck 1 was loaded with background traffic from Server 1 to Client 1, and bottleneck 2 with traffic from Server 2 to Client 2. The link characteristics for WLAN and 3G links are set as follows.

- WLAN: Capacity=25 Mbit/s, Delay=25 ms, Loss=1%
- 3G: Capacity=5 Mbit/s, Delay=65 ms, Loss=0%

Based on measurements carried in real networks, the queue lengths at each router interface were set to 100 packets for WLAN and 3750 packets for 3G. The losses applied to the WLAN path are random.

1) Network and System characteristics: System settings are known to impact TCP's performance. In order to emulate realistic network scenarios, we used system settings close to the standard characteristics of each technologies. The TCP buffer sizes (send/receive) were set to be equivalent to widely known Android settings, that are configured as follows.

- Homogeneous (WLAN): 1024 KiB/2048 KiB.
- Heterogeneous (3G+WLAN): 1024 KiB/2048 KiB.

For bulk traffic experiments, we set both send and receive buffers to 16 MiB to evaluate MPTCP's aggregation capability.

To ensure independence between runs, the cached TCP values were flushed after every run. We focused on congestion avoidance; therefore, we discarded the initial phase for each experiment and analyzed a period of 90 s for bulk and constant bitrate (CBR) traffic. For single-path TCP flows, we used TCP-Reno, therefore, fairly compairing against MPTCP-OLIA.[3]

2) Application Traffic: We considered three different types.

a) Video Streaming: We considered constant bit-rate (CBR) video traffic with a frame size of 5 KiB on the application level and a rate of 1 Mbps. This is in line with the recent measurement studies [17] showing that more than 53% of the downstream traffic in North America is video streaming, and with other reports [18] predicting further increase,

b) Web Traffic: We selected three websites of different sizes, small, medium and large (see Table I), as a good set of typical website sizes. To mimic the behavior of a real browser downloads were performed with 6 concurrent connections.

c) Bulk Transfer: We completed the evaluation with the most common case for MPTCP — a buk transfer, of 64 MiB.

[3]TCP-Linux kernel 3.14.33 is used throughout our evaluations.

3) Background Traffic: A synthetic mix of TCP and UDP traffic was generated with D-ITG [19] as background traffic in order to create a realistic environment. The TCP traffic was composed of saturated sender and rate-limited TCP flows with a exponentially distributed mean rate of 157 pps. The UDP traffic was composed of UDP on/off flows with Pareto distributed on and exponentially distributed off times. Each flow has an exponentially distributed mean rate of 100 kbps in the heterogeneous scenario and 500 kbps in the homogeneous scenario. Packet sizes were varied with a mean of 1000 Bytes and RTTs between 20 and 100 ms. We repeated all experiment settings 50 times, in both emulation and real scenarios.

IV. Scheduling Against HoL-Blocking

In the following, we discuss both Delay-Aware Packet Scheduler (DAPS) [7], [8] and Out-of-order Transmission for In-order Arrival Scheduler (OTIAS) [9], evaluating them in common scenarios, and commenting on their implementation.

A. Delay-Aware Packet Scheduler (DAPS)

The DAPS algorithm was proposed in two versions. In [7], it pursues the goal to make segments arrive in order by planning which subflows the next segments should be sent over based on both the forward delay and CWND of each subflow. A schedule is created to span the least common multiple (LCM) of the forward delays $\mathrm{lcm}(D_i \in \{D_1, D_2, \ldots, D_n\})$. Algorithm 1 shows the main loop of the mechanism.

As an example, assume two subflows with similar capacities, but with a subflow having a forward delay ten times higher than the fast subflow. DAPS will derive the following schedule: segments 1...10 will be sent on the fast subflow, and segment 11 on the other subflow. Ideally, segment 11 will arrive right after segment 10, thereby avoiding HoL-blocking.

In [8], DAPS is formulated for a scenario with only two subflows (r_s and r_f). It is also a simplification of the original algorithm [7] as it does not take CWND asymmetry into account, only considering the subflows' RTT ratio (η) and the CWND of the fast subflow.

Since both algorithms are comparable, we consider only the original DAPS [7] in our evaluations. We ignore the simplifications presented in [8], as they were only introduced to ease the implementation in the ns-2 of CMT-SCTP.

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

The OTIAS algorithm [9] is based on the idea of scheduling more segments on a subflow than what it can currently send. Queues may therefore build up at each subflow of the sender, under the assumption that these segments will be sent as soon

Table I
Web Traffic Generation

Domain name	Number of Objects	Size of Objects
http://www.wikipedia.org	15	72 KiB
http://www.amazon.com	54	1024 KiB
http://www.huffingtonpost.com	138	3994 KiB

Algorithm 1 DAPS [7]

$S_{max} \leftarrow 0$
for $P_i \in \{P_1, P_2, ..., P_n\}$ **do**
  $SEQ_{P_i} \leftarrow InitializeVector()$
end for
for $P_i \in \{O_1, O_2, ..., O_{\sum_i \in 1,2,...,n \frac{lcm(D_i)}{D_i}}\}$ **do**
  $SEQ_{P_i} \leftarrow Append(SEQ_{P_i}[S_{max} + 1, S_{max} + C_i]$
end for
$t \leftarrow 0$
while $t < \text{lcm}(D_i \in \{D_1, D_2, ..., D_n\})$ **do**
  for $P_i \in \{P_1, P_2, ..., P_n\}$ **do**
    if $t \equiv 0 \ (mod \ D_i)$ **then**
      $Transmit(P_i, SEQ_{P_i}[\frac{t}{D_i}])$
      $S_{max} \leftarrow S_{max} + C_i$
    end if
  end for
  $t \leftarrow t + 1$
end while

Where:

- $\{P_1, P_2, ..., P_n\}$ set of paths
- $\{D_1, D_2, ..., D_n\}$ paths' respective forward delays
- SEQ_{P_i} seqnos of packets to be transmitted on P_i

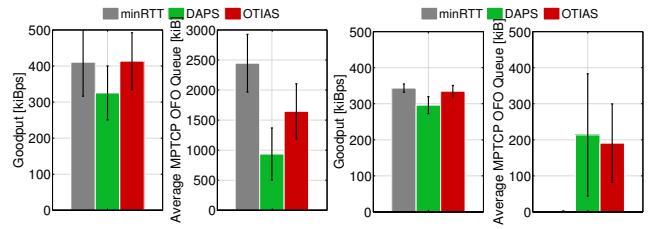
as there is space in the CWND for the subflow. When asked to schedule a new segment, the algorithm estimates its arrival time if sent over each subflow (T_i^j), and chooses the subflow with the earliest arrival time. The estimation is performed based on a subflow's RTT, its CWND, the number of in-flight packets and the number of already queued packets. If there is space in the CWND, the segment would be sent immediately, yielding an arrival time of approximately $RTT/2$ (assuming symmetric forward and backward delays). If the CWND is full, however, the segments will have to wait in the subflow's queue. Assuming a send rate of 1 CWND per RTT, the additional waiting time is calculated as $\texttt{RTT_to_wait}_i^j$. Algorithm 2 shows the main loop of the OTIAS mechanism.

Algorithm 2 OTIAS [9]

for `each available subflow` j **do**
  $\texttt{pkt_can_be_sent}_j = cwnd_j - unacked_j$
  $\texttt{RTT_to_wait}_i^j = \left\lfloor \frac{\texttt{not_yet_sent}_j - \texttt{pkt_can_be_sent}_j}{cwnd_j} \right\rfloor$
  $T_i^j = (\texttt{RTT_to_wait}_i^j + 0.5) \times srtt_j$
  if $T_i^j < min_T$ **then**
    $min_T = T_i^j$
    $\texttt{selected_subflow} = j$
  end if
end for

C. Comparative evaluation of DAPS and OTIAS

Although DAPS and OTIAS have the same goal to reduce HoL-blocking, they follow different approaches: DAPS creates a schedule for the distribution of future segments into the available subflows for a *scheduling run* and follows this schedule until it is completed, after which planning for the next run is determined. On the other hand, OTIAS decides which subflow to use on a *per-packet* basis. It takes into account the RTTs and the queue sizes of the subflows at a given moment and it is closer to MPTCP's default scheduler (minRTT) in this respect, albeit taking into account more information from the subflows.



(a) **3G+WLAN** (b) **WLAN+WLAN**

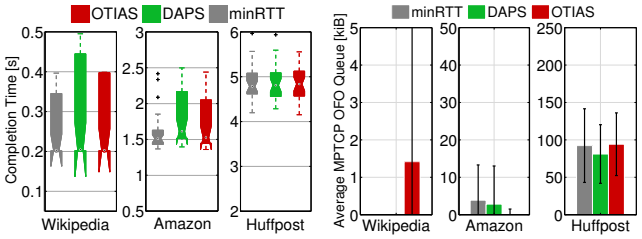
Figure 3. Goodput and OFO queue for bulk traffic between DAPS, OTIAS and minRTT.

OTIAS operates based on current data and is able to react more dynamically to network changes, where DAPS can only react to changes in the next scheduling run. OTIAS is however still less dynamic than MPTCP's minRTT since it builds up queues on the subflows. If a segment that had already been sent is blocking the connection, *e.g.*, it could be delayed or lost, the queued packets would linger at the sender more than assumed, disturbing the created schedule. Moreover, MPTCP's default scheduler retransmission mechanism, retransmitting a packet on the fastest subflow [4], is not applicable if a send queue exists for a subflow, as that segment would have to wait in the queue before retransmission.

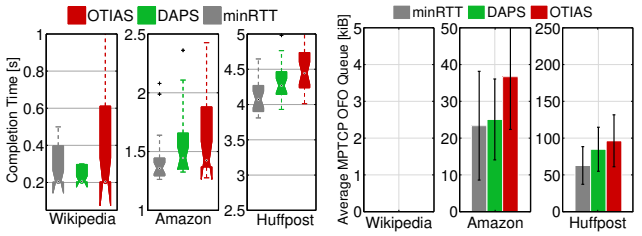
In the following we present an evaluation of DAPS and OTIAS against MPTCP's minRTT with bulk, web and CBR traffic through emulations. We look at application goodput for bulk transfers, completion times for web transfers, and average application delay for CBR traffic. In all cases, we also sample the maximum value of the out-of-order (OFO) queue every 10 ms during the experiments and present the results.

1) Bulk: Figure 3 shows DAPS, OTIAS and MPTCP's default scheduler goodput and OFO buffer size for bulk transfer in both 3G+WLAN and WLAN+WLAN scenarios. OTIAS provides a goodput increase of 6% but requires 35% less OFO buffer compared to MPTCP's minRTT. On the other hand, DAPS provides a goodput decrease of 27% and requires 65% less OFO buffer compared to MPTCP's default scheduler. In WLAN+WLAN scenarios, MPTCP's default scheduler has a 3.5% lower goodput compared to OTIAS, which on the contrary takes about 87% more OFO buffer. DAPS delivers goodput values of about 16% less compared to MPTCP's default scheduler with about 97% more OFO buffer.

2) Web: Figure 4 shows the completion times and OFO buffer sizes for DAPS, OTIAS and MPTCP's default schedulers in both 3G+WLAN and WLAN+WLAN scenarios. For 3G+WLAN, in Figure 4(a), all scheduler algorithms perform similarly in terms of completion time. However, for larger object sizes, we observe a larger OFO buffer size. In WLAN+WLAN, in Figure 4(c), DAPS and OTIAS struggle



(a) **3G+WLAN**, Completion time (b) **3G+WLAN**, OFO queue



(c) **WLAN+WLAN**, Completion time(d) **WLAN+WLAN**, OFO queue

Figure 4. Completion time and OFO queue for web traffic (Wikipedia, Amazon and Huffington Post) for DAPS, OTIAS and minRTT.

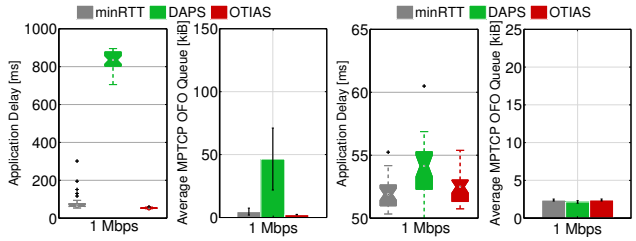
when both paths have higher loss rates, because DAPS cannot react quickly enough to changes on the paths, and OTIAS builds queues that also don't allow immediate reaction. While the losses on the WLAN paths cause higher OFO buffer size in WLAN+WLAN, the path heterogeneity is the main reason for the higher OFO size in 3G+WLAN.

3) CBR: Figure 5 shows the average application delay and the OFO buffer size for DAPS, OTIAS and MPTCP's default schedulers in both 3G+WLAN and WLAN+WLAN scenarios. Both 3G+WLAN and WLAN+WLAN yield higher application delay with DAPS. OTIAS can reduce the usage of the 3G subflow in the 3G+WLAN scenario, leading to improved application delay compared to MPTCP's default scheduler. However, for the WLAN+WLAN scenario, OTIAS provides higher delay values compared to MPTCP due to the lack of design for a reinjection mechanism. Moreover, MPTCP's default scheduler PR mechanism can partially overcome path heterogeneity in 3G+WLAN, where we can observe *burst* of packets on the 3G path, which lead to spikes in the OFO buffer, resulting in higher application delay.

D. Successes and Failures of Existing Algorithms

Overall, we observe that, although all state-of-the-art approaches address the challenges of multipath scheduling in heterogeneous scenarios, trying to overcome receive-window limitation and, consequently, HoL-blocking, they still fail in some typical use-case scenarios settings, *e.g.*, heterogeneous delays and/or loss rates, as well as with excessive delays due to buffering. Here, we comment on the strong and weak aspects of the state-of-the-art proposals just evaluated.

1) OTIAS: Although OTIAS can make decisions on a per-packet basis (subflow j and packet i) reacting fast and using current state from the network ($cwnd_j$ loop), it builds up queues on the subflows with lowest RTTs, regardless of their CWND state, *i.e.*, it does not restrict the scheduler if the



(a) **3G+WLAN** (b) **WLAN+WLAN**

Figure 5. Packet delay and OFO queue for CBR traffic for DAPS, OTIAS and minRTT.

CWND is full. In addition, the algorithm assumes symmetric forward delays ($OWD = RTT/2$), and scheduler reinjections (retransmissions) are not mentioned. While OTIAS can yield good results with heterogeneous RTTs, if the heterogeneity is too large and losses occur in one of the subflows, the algorithm will build up long queues in the subflows with lower RTTs, reducing their ability to overcome HoL-blocking. In homogeneous scenarios the OTIAS scheduler delivers lower performance due to not using both subflows as fully as MPTCP's default scheduler.

2) DAPS: The DAPS implementation is more complex, requiring more memory at run-time to keep the schedule run. Furthermore, DAPS is not able to react upon network changes in a timely manner due to long schedules arising from high heterogeneity in the subflow delays, *i.e.*, high LCM in Algorithm 1. Last but not least, DAPS will use all subflows that can send, even if a certain subflow's contribution is very low. This is the main contrast compared to both OTIAS and MPTCP's default schedulers, which can reduce the slow subflow contribution, if a faster subflow can sustain the required rate. This is particularly important for transfers where the sender is not saturated. Finally, similar to OTIAS, DAPS does not have a defined behaviour for scheduler reinjections.

V. BLEST: Blocking Estimation-based MPTCP Scheduler

Based on the observations from Section IV, we introduce a new algorithm, BLEST, addressing the challenges of reducing HoL-blocking, spurious retransmissions, and hence increasing application performance in heterogeneous scenarios. The scheduling is based on a new metric, estimating the amount of HoL-blocking, which might result from scheduling a packet on a give subflow.

For each new segment, MPTCP's default scheduler, minRTT, chooses the subflow with lowest RTT among all subflows ready to send, *i.e.*, with space in the CWND. When MPTCP detects that it cannot send new data due to a full send window (mirror of receive window at the sender), it will resend the segment blocking the fastest subflow, but only if it hasn't been sent on that subflow before. It will also penalise the slow subflow responsible for blocking, halving its CWND. The idea is to reduce its contribution preventing further HoL-blocking. Such an approach reduces the chance of HoL-blocking only for a limited amount of time. In other words,

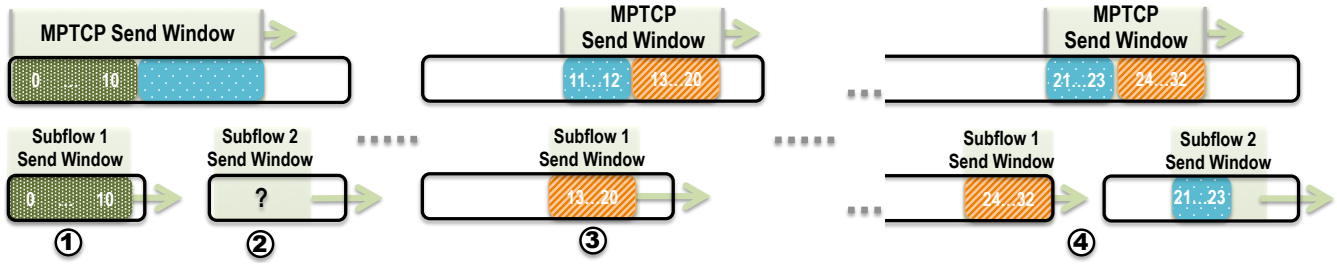


Figure 6. MPTCP example with BLEST: In ①, segments 0. . . 10 are in flight on subflow 1, the subflow with lowest delay. In ② it is uncertain how many segments should be sent on subflow 2, which has a higher delay. While subflow 2's window could accommodate more data, only segments 11. . . 12 are allocated, due to BLEST's blocking prediction. Here, minRTT would allocate as much data as fits into subflow 2's window given its CWND. In ③ subflow 1 can advance with segments 13. . . 20, because 0. . . 10 were acknowledged. At ④ both subflows can advance with MPTCP's send window with subflow 1 carrying segments 24. . . 32 and subflow 2 carrying 21. . . 23.

after the CWND was reduced by penalisation, the congestion control will start increasing it again, until a recurrence of blocking. Furthermore, the approach is reactive as it depends on blocking to trigger PR at the sender. The PR mechanism itself is detrimental in the long run, since it keeps the CWND of slow subflow artificially low.

To overcome the issues of the PR, we propose a proactive scheduler where we decide at packet scheduling time whether to send packets over the slow subflow or not. The decision is based on MPTCP's send window. MPTCP maintains a send window on its control-plane for each MPTCP connection, one level above the subflows. This window is necessary due to the full multiplexing among all subflows belonging to the same MPTCP connection. However, due to its scheduler design, if data is not acknowledged in one of the subflows, MPTCP's send window can be temporarily blocked, stalling the whole multipath connection.

BLEST assumes that a segment will occupy space in MPTCP's send window ($MPTCP_{\text{SW}}$) for at least RTT_S if it is sent now on subflow S, as illustrated in Figure 6. 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, as these segments can be acknowledged earlier. The remaining send window can be used by the faster subflow (*i.e.*, lower RTT subflow), F. This means that HoL-blocking would occur if F were not able to send 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 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

$$rtts = RTT_S/RTT_F$$
$$X = MSS_F \cdot (CWND + (rtts - 1)/2) \cdot rtts$$

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

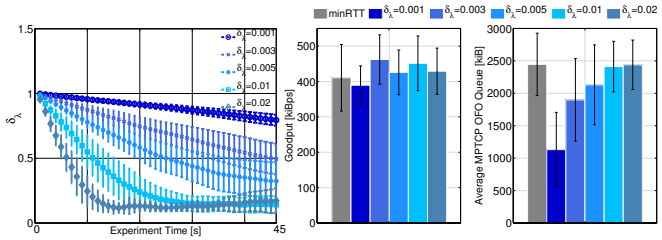


Figure 7. **3G+WLAN** and BLEST's λ parameter influence on bulk traffic with varying δ_λ=0.001, 0.003, 0.005, 0.01, and 0.02; compared against minRTT.

The estimate of X, however, can be inaccurate at times. To address this, we introduce a correction factor λ, to scale X. λ is adjusted as follows. HoL-blocking during one RTT_F is an event that triggers an increase of λ by δ_λ; the absence of HoL-blocking triggers a decrease by δ_λ. In the beginning of the connection we set λ=1.0, *i.e.*, no correction of the estimation.

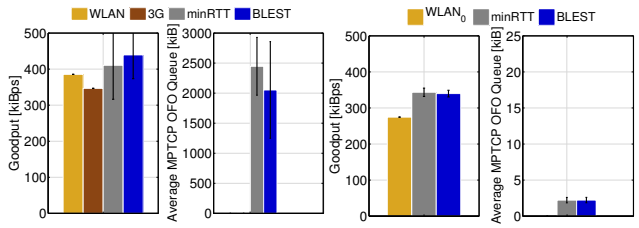
Figure 7 shows how λ changes over time in our scenario with different δ_λ. With $\delta_\lambda = 0.001$ we see that λ changes slowly towards a value that represents the reality on the (lossy WLAN) link. Note that X is over-estimated in the beginning of the transfer. Therefore, most of the traffic is sent over the WLAN link leading to a reduced goodput. However, in time, the estimate is corrected by λ to reach a steady value where the HoL-blocking is minimised.

On the left side, Figure 7 shows the first 45 seconds of a bulk transfer and how λ corrects the estimation (each dot in the plot curves show the average and standard deviation over 1s) of the rate of the faster subflow throughout the period. On the right side, Figure 7 shows the effect in the OFO buffer size and, consequently, in the goodput for different δ_λ. λ is corrected to lower values than its initial setting of 1.0, because the model does not incorporate losses.

VI. Evaluation

One of MPTCP's goals is to perform at least as well as TCP on the best path. For this reason, we compare MPTCP's default scheduler, minRTT, and BLEST against single path TCP on 3G and WLAN paths. We include both 3G+WLAN and WLAN+WLAN scenarios in our evaluation to illustrate the improvements in heterogeneous settings, while not impacting MPTCP in homogeneous scenarios. In the following we show emulations and real network experiments results.

ISBN 978-3-901882-84-5, IFIP Networking 2016 © 2016 IFIP



(a) 3G+WLAN (b) WLAN+WLAN

Figure 8. **3G+WLAN** and **WLAN+WLAN** scenarios for bulk traffic with minRTT, BLEST and TCP on 3G and WLAN.

A. Emulation Experiments

1) Bulk: Increasing application goodput for bulk transfer has been one of the most common ways to evaluate MPTCP's performance. Figures 8(a) and 8(b) compares the performance in terms of goodput and OFO queue size for minRTT and BLEST with bulk traffic in 3G+WLAN and WLAN+WLAN scenarios, respectively. In 3G+WLAN, we observe that BLEST reduces OFO buffer size by 19%, while it increases application goodput by 12%. Note that the MPTCP default scheduler's *penalisation and retransmission* (PR) mechanism has a particular negative impact in 3G+WLAN. As illustrated in Table II, MPTCP's PR mechanism can send up to 0.53 MiB retransmissions, to overcome blocking of the WLAN path. BLEST achieves better aggregation with less OFO buffer, saving up to 80% of retransmissions. In WLAN+WLAN, BLEST achieves similar application goodput with negligible OFO buffer size of 2.5 kiB compared to minRTT.

2) Web: The total download time is not a perfect metric as most browsers start rendering the page before the transmission is complete. However, we are focusing on the transport-level performance, and discard any browser-related optimisations. Figures 9(a) and 10(a) show the completion times for minRTT and BLEST for web traffic with different object sizes, see Table I. We also compare the OFO buffer size shown in Figures 9(b) and 10(b), and quantify the contribution of the additional subflow with smaller web objects, the amount of bytes transferred on each subflow relative to the transfer size, see Figures 9(c) and 10(c). In 3G+WLAN, for smaller web objects such as Wikipedia, the contribution of the additional subflow (3G) can be considered negligible, with only up to 2% of the total transfer. However, the small contribution of the 3G path for Amazon can cause an impact of up to 7% reduction in the completion time for BLEST compared to minRTT, see Table III and Figure 9(b). For Huffington Post, although the contribution of the additional subflow is still comparably low (about 2%), the completion time for BLEST is 6% lower than minRTT. In WLAN+WLAN, BLEST provides an improvement of 3% for Huffington Post and 2% for Amazon in completion times compared to minRTT. Overall, Table III illustrates the benefits of BLEST where the lowest completion time is achieved by the proposed BLEST algorithm for both heterogeneous and homogeneous scenarios for all the websites evaluated.

3) CBR: Live video has higher requirements of low latency compared to other forms of video streaming, *e.g.*, video on demand. Moreover, live video is more sensitive to network delay variations and, therefore, impacts the user experience the most. As we want to assess whether MPTCP could be used for applications other than bulk traffic, we evaluate live video performance that is more sensitive to latency. Figures 11(a) and 11(b) show the average application delay for minRTT and BLEST for CBR traffic with 1 Mbps. In the 3G+WLAN scenario, BLEST improved the application delay over minRTT by 8% for CBR (1 Mbps) and a slight improvement in OFO buffer size of 8% is also achieved, see also Table 11. In the same scenario and with the same application traffic, comparing BLEST to results shown in Figures 4, BLEST performed worse than OTIAS with CBR, because OTIAS completely discarded the 3G path. In contrast to that, DAPS keeps utilising the 3G path. In WLAN+WLAN shown in Figure 11(b), BLEST performed similar to MPTCP's default scheduler as expected.

B. Real Experiments

Finally, we validate the performance of the different schedulers with real-network experiments within the same topology as shown in Figure 2 for the emulation experiments, but now constructed over NorNet [20]. To generate background traffic, we use Virtual Machines (VM) from five commercial cloud service providers (2x in Europe, 1x in North America and 2x in Asia) connected via 100 Mbps links, as described in Section III, towards the server machine in Figure 12. We also use consumer hardware with a RaspberryPi connected to a home DSL provider via WLAN and another interface via 3G/3.5G to a mobile broadband operator. On the RaspberryPi side, background traffic from other connected devices congested both WLAN and 3G. The experimental setup is shown in Figure 12.

Table II
PENALISATION AND RETRANSMISSION MECHANISM TRIGGER IN 3G+WLAN WITH BULK TRAFFIC SHOWN IN FIGURE 3

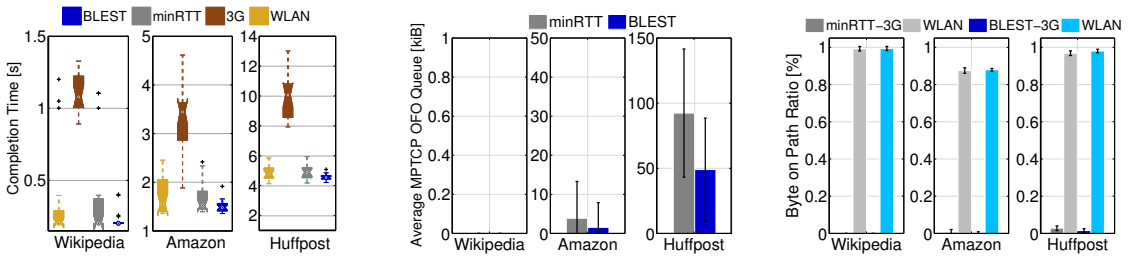
	Scheduler	Traffic	Retrans. Packets
3G+WLAN	**minRTT**	Bulk	366.37 **0.53 MiB**
	BLEST		70.3 **0.1 MiB**

Table III
AVERAGE WEB COMPLETION TIME, SEE FIGURES 4, 9, AND 10

			minRTT	OTIAS	DAPS	BLEST
Scenario	**Traffic**		**Completion Time [s]**			
3G+WLAN	Web	Wikipedia	0.421	0.392	0.435	**0.337**
		Amazon	1.60	1.724	1.789	**1.503**
		Huffington Post	4.87	4.858	4.932	**4.62**
WLAN+WLAN	Web	Wikipedia	0.398	0.4107	0.333	**0.324**
		Amazon	1.461	1.621	1.598	**1.456**
		Huffington Post	4.218	4.509	4.393	**4.114**

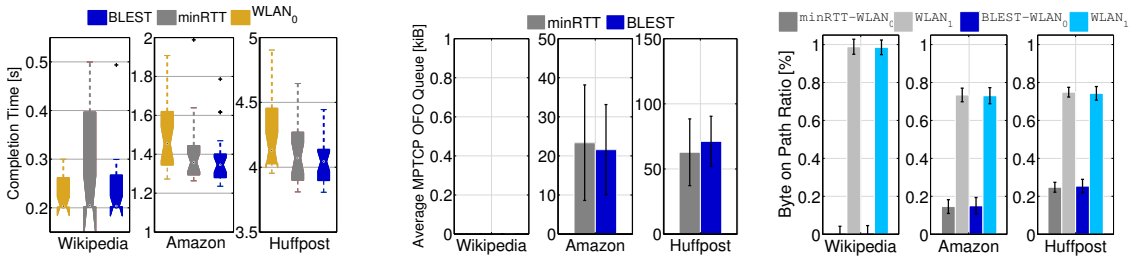
Table IV
AVERAGE CBR APPLICATION DELAY, SEE FIGURES 5AND 11

			minRTT	OTIAS	DAPS	BLEST
Scenario	**Traffic [Mbps]**		**Application Delay [ms]**			
3G+WLAN	CBR	1	68	53.2	843.7	**62.8**
WLAN+WLAN	CBR	1	52.18	53.49	54.02	**52.24**



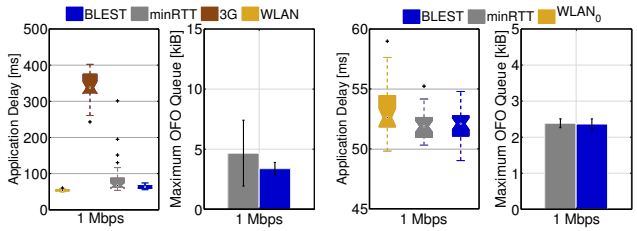
(a) Completion Time (b) MPTCP OFO Queue (c) Byte on Path: 3G and WLAN

Figure 9. **3G+WLAN** for web traffic with Wikipedia, Amazon and Huffington Post with minRTT, BLEST and TCP on 3G and WLAN.



(a) Completion Time (b) MPTCP OFO Queue (c) Byte on Path: $WLAN_0$ and $WLAN_1$

Figure 10. **WLAN+WLAN** for web traffic with Wikipedia, Amazon and Huffington Post with minRTT, BLEST and TCP on WLAN ($WLAN_0$ and $WLAN_1$).



(a) 3G+WLAN (b) WLAN+WLAN

Figure 11. **3G+WLAN** and **WLAN+WLAN** scenarios for 1 Mbps CBR traffic with minRTT, BLEST and TCP on 3G and WLAN.

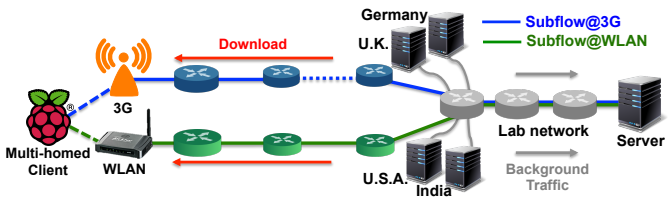
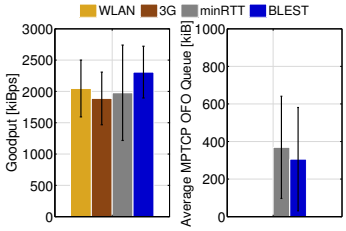


Figure 12. Real network experiment setup



(a) 3G+WLAN

Figure 13. **3G+WLAN** for bulk traffic in real experiments, see Figure 12.

In our experiments, we used the same parameters and settings from Section III as well as the same traffic from Section VI-A. We evaluate the performance of different schedulers under realistic network conditions, with real-network experiments, in a constructed non-shared bottleneck scenario as used in the emulation experiments shown in Figure 2.

1) Bulk: Figure 13 shows the application goodput and OFO buffer size for bulk traffic with minRTT and BLEST compared to TCP on 3G and WLAN paths. BLEST achieves on average 18% higher application goodput aggregation, while reducing the amount of retransmissions by more than 37%, see Table V, with a slight improvement in OFO buffer size of 3%.

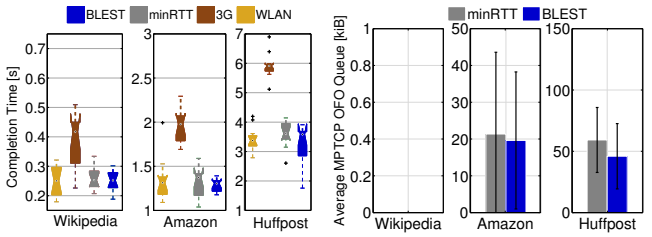
2) Web: Figures 14(a) and 14(b) show the completion times and OFO buffer sizes for the web transfers. With larger object sizes, BLEST reduces the completion time by up to 10%, while reducing the OFO size by up to 25%. Thus, MPTCP's performance with BLEST is closer to the WLAN path, only 3% worse than TCP on the best path (WLAN).

3) CBR: Figure 15(a) shows the the average application delay and OFO buffer size for the 1 Mbps CBR traffic with both minRTT and BLEST. BLEST improves the application delay by 11% while reducing the OFO size by more than 20%. We noticed, looking at single experiments, that MPTCP's default scheduler had small packet *bursts* sent over 3G, causing some spikes in the OFO buffer and, consequently, increasing the application delay. However, BLEST used the 3G path in the majority of the cases to send few single packets.

VII. Conclusion

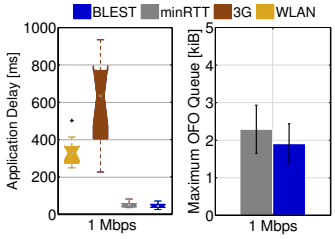
Path heterogeneity is rather the rule than the exception with MPTCP. Even subflows from a single machine can follow different paths to the destination with distinct delay, capacity,

ISBN 978-3-901882-84-5, IFIP Networking 2016 © 2016 IFIP



(a) Completion time (b) MPTCP OFO queue

Figure 14. **3G+WLAN** for web traffic with Wikipedia, Amazon and Huffington Post in real experiments, see Figure 12.



(a) Delay and MPTCP OFO queue

Figure 15. **3G+WLAN** for CBR traffic in real experiments, see Figure 12.

and losses. Such path heterogeneity results in HoL-blocking at the receiver undermining MPTCP's overall performance. To overcome path heterogeneity, MPTCP follows a reactive approach and penalizes the subflows that cause HoL-blocking, through the *penalisation and retransmission* mechanism.

In this paper, we highlighted the limitations of such an approach for different application types in heterogeneous scenario through emulations and real-world experiments. Moreover, we have implemented and systematically evaluated scheduling algorithms aiming at mitigating this issue. We found, however, that neither was able to perform well in all multi-homing scenarios and traffic use-cases. We therefore proposed BLEST, a new scheduler based on a BLocking time ESTimation. Compared to previous proposals, BLEST directly considers the prospective HoL-blocking as a metric to minimise, and based on this metric, it dynamically selects whether it is worthwhile to schedule a packet on a specific subflow, or to ignore it. This allowed us to eliminate the *penalisation and retransmission* by implementing a more robust, proactive, scheduling metric. We evaluated our algorithm in emulated and real experiments with different application traffic (CBR, web and bulk). By comparing BLEST with minRTT, as well as the alternative DAPS and OTIAS, we showed that our approach outperfoms all algorithms across the presented scenarios, achieving its goal of reducing HoL-blocking, and consequently unnecessary retransmissions. This results in an increasing application goodput, lower packet delay and completion time, and reduced receiver buffer size.

Table V
PENALISATION AND RETRANSMISSION ALGORITHM RETRANSMISSIONS' OVERHEAD IN 3G+WLAN WITH BULK TRAFFIC SHOWN IN FIGURE 12

	Scheduler	Traffic	Retrans. Packets
3G+WLAN	**minRTT**	Bulk	33.42
	BLEST		21.3

For future work, we believe that both BLEST and OTIAS follow the right approach towards robust and effective scheduling for heterogeneous scenarios. We want to expand our evaluation with the method proposed in [5], add other elements of heterogeneity, *e.g.*, other network access technologies, evaluate different application performance metrics, *e.g.*, throughput aggregation versus delay constraints, increase the number of subflows and test the approach in mobility scenarios.

REFERENCES

[1] G. Sarwar, R. Boreli, E. Lochin, and A. Mifdaoui, "Performance evaluation of multipath transport protocol in asymmetric heterogeneous network environment," in *ISCIT 2012*.

[2] C. Raiciu, C. Paasch, S. Barré, A. Ford, M. Honda, F. Duchêne, O. Bonaventure, and M. Handley, "How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP," in *NSDI 2012*.

[3] S. Ferlin, T. Dreibholz, and O. Alay, "Multi-path transport over heterogeneous wireless networks: Does it really pay off?" in *GLOBECOM 2014*.

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

[5] C. Paasch, "Improving multipath TCP," Ph.D. dissertation, UCLouvain / ICTEAM / EPL, Nov. 2014.

[6] A. Singh, C. Goerg, A. Timm-Giel, M. Scharf, and T.-R. Banniza, "Performance comparison of scheduling algorithms for multipath transfer," in *GLOBECOM 2012*.

[7] 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 *WAINA 2013*.

[8] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent delay-aware packet scheduling for multipath transport," in *ICC 2014*.

[9] F. Yang, Q. Wang, and P. Amer, "Out-of-order transmission for in-order arrival scheduling policy for multipath TCP," in *WAINA 2014*.

[10] B. Arzani, A. Gurney, S. Cheng, R. Guerin, and B. T. Loo, "Impact of path characteristics and scheduling policies on MPTCP performance," in *WAINA 2014*.

[11] S. Barré, C. Paasch, and O. Bonaventure, "Multipath TCP: From theory to practice," in *IFIP Networking 2011*.

[12] C. Paasch, G. Detal, F. Duchêne, C. Raiciu, and O. Bonaventure, "Exploring mobile/WiFi handover with multipath TCP," in *CellNet 2012*.

[13] C. Paasch, R. Khalili, and O. Bonaventure, "On the benefits of applying experimental design to improve multipath TCP," in *CoNEXT 2013*.

[14] C. Raiciu, S. Barré, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," in *SIGCOMM 2011*, Toronto, ON, Canada.

[15] I. A. Halepoto, F. Lau, and Z. Niu, "Management of buffer space for the concurrent multipath transfer over dissimilar paths," in *DINWC 2015*.

[16] J. Ahrenholz, "Comparison of CORE network emulation platforms," in *MILCOM 2010*.

[17] Sandvine Intelligent Broadband Networks, "Global Internet Phenomena Report," Jul. 2013. [Online]. Available: https://web.archive.org/web/20141216103806/https://www.sandvine.com/downloads/general/global-internet-phenomena/2013/sandvine-global-internet-phenomena-report-1h-2013.pdf

[18] Cisco Visual Networking Index, "Forecast and Methodology, 2014–2019," 2014. [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.pdf

[19] A. Botta, A. Dainotti, and A. Pescapé, "A tool for the generation of realistic network workload for emerging networking scenarios," *Computer Networks*, vol. 56, no. 15, pp. 3531–3547, 2012.

[20] E. G. Gran, T. Dreibholz, and A. Kvalbein, "NorNet core — a multi-homed research testbed," *Computer Networks, Special Issue on Future Internet Testbeds*, vol. 61, pp. 75–87, Mar. 2014.