LISA: A Linked Slow-Start Algorithm for MPTCP

Runa Barik*, Michael Welzl*, Simone Ferlin†, Ozgu Alay†
*IFI, UiO, Norway
{runabk, michawe}@ifi.uio.no

†Simula Research Laboratory, Norway
{ferlin, ozgu}@simula.no

***Abstract*—One of the main goals of multipath TCP (MPTCP) is to achieve higher throughput than regular TCP by utilizing multiple paths simultaneously. When these paths share a common bottleneck, MPTCP tries not to be more aggressive than a regular TCP flow. This is achieved by MPTCP's coupled congestion control mechanism that couples the increase factor of MPTCP's subflows in congestion avoidance. However, slow-start remains unchanged and behaves uncoupled for each subflow, affecting MPTCP and concurrent traffic at the bottleneck. We propose LISA, a simple algorithm for coupling MPTCP subflows in slow-start, and investigate the trade-off that this coupling entails. Our evaluations show that coupling in slow-start not only provides gains for MPTCP but also for a concurrent TCP at the bottleneck.**

***Index Terms*—MPTCP, slow-start, initial window, fairness**

I. Introduction

The Internet was much simpler when the Transmission Control Protocol (TCP) was first designed 30 years ago. At that time, end-hosts had a single interface, therefore, TCP was built around the notion of a (single) connection between two hosts. However, today's networks paint a very different picture where server machines are multi-homed and end-hosts are multiple connected. For instance, data center networks have a large redundant infrastructure with many paths between any two servers. Similarly, smartphones are equipped with two interfaces: a WiFi and a mobile broadband (e.g., 3G/4G). Although today's networks provide many possible paths between end-hosts, any given TCP session will use only one of them.

Multipath TCP (MPTCP) is an ongoing effort of the Internet Engineering Task Force's (IETF) Multipath TCP working group that aims to extend TCP by allowing multiple paths to be used simultaneously to maximize resource usage and increase reliability [9]. MPTCP uses TCP options for signaling, hence, from the network's perspective, it looks like regular TCP, making it deployable in today's Internet. MPTCP can efficiently pool network's resources by presenting a regular TCP socket to the application, which underneath leverages multiple subflows and it is able to multiplex a single data stream across them.

One of MPTCP's main goals is to guarantee fairness to regular TCP at shared bottlenecks. To address this, coupled congestion control algorithms are proposed for MPTCP [12], [15], [14], [10]. However, the current MPTCP congestion control algorithms only consider fairness in congestion avoidance (CA). Today, most of the TCP sessions in the Internet constitute short flows (e.g. web requests) [6] and more than 40% of the web transfers are of size smaller than 1MB [3]. For these short flows, TCP will likely never leave the slow-start (SS) phase, therefore SS behaviour becomes of critical importance for the performance. However, in the current MPTCP's congestion control implementations, each subflow in SS behaves as an independent TCP connection. In SS, following default Linux, each subflow's initial window (IW) starts with 10 segments [7] and it exponentially increases [9]. For an MPTCP connection with 2 subflows, this results in MPTCP doubling its queue occupancy, and hence, increasing the unfairness compared to a concurrent TCP flow at the shared bottleneck. This further escalates with an increasing number of subflows.

In this paper, we focus on MPTCP's slow-start. We first illustrate that MPTCP's uncoupled SS harms the performance of both MPTCP and the concurrent TCP at the shared bottleneck. To address this, we propose a linked slow-start algorithm (LISA) where MPTCP subflows are coupled during the SS. Our results show that LISA improves the MPTCP transfer completion time by reducing the number of retransmission and reduces MPTCP's negative effect on competing TCP traffic.

II. Problem statement

MPTCP's path-manager defines how subflows are added to an existing connection. The default path-manager *fullmesh* opens a full mesh of subflows between all IP addresses of the end-hosts. The remote end-host advertises all additional IP addresses to the connection initiator within the same RTT [11]. Each of these subflows is a regular TCP connection (i.e., they have connection establishment with a three-way handshake, data transmission with slow-start and congestion avoidance and connection termination phases) [9]; thus, after 3 RTTs, all new subflows can start sending, each of them with IW10.

According to [9], and in all current implementations of different MPTCP's coupled congestion control algorithms [10],[14], MPTCP subflows in slow-start are uncoupled, i.e., each behaving as regular TCP with IW10 and increasing by 1 after every ACK. Therefore, in slow-start, all subflows independently double their congestion windows ($cwnd$) as in regular TCP [9], resulting in also doubling MPTCP's *compound* $cwnd$. Although coupling slow-start could seem unnecessary, considering that each new subflow starts with IW=10[1]

[1]Default Linux kernel 3.14.22 with MPTCP v0.89.3

and assuming that there are existing subflows still in slow-start, the compound MPTCP $cwnd$ can briefly increase by a large number when new subflows join.

We illustrate this behavior with an example: Fig. 1(a) shows the $cwnd$ evolution of a 300 KB file transfer across a 2.5 Mbps shared bottleneck with TCP (CUBIC) and MPTCP (see the measurement setup in Section IV). Here, we observe that TCP completes the transmission approximately 50ms earlier than MPTCP. This is due to the large overshoot when the second subflow joins, causing more retransmissions as shown in Fig. 1(b). In order to eliminate this behavior, we propose a **LI**nked **S**low-Start **A**lgorithm (**LISA**) for MPTCP.

III. LISA

The idea behind LISA is that each new subflow takes a *credit* from an existing subflow needed for its own IW, hence linking the subflow's congestion windows in slow start. Doing so, it has 10 packets as upper limit based on [7] and 3 packets as lower limit based on [4][2]. The design choice of bounding the IW between 3 and 10 is based on the RFC standards [4], [7] and the main reason behind it is to let a subflow compete reasonably when it is not sharing a bottleneck with other subflows. We also divide the $cwnd$ fairly in order to give all subflows an equal chance to compete with background traffic.

The LISA algorithm, whose pseudo-code is presented in Algorithm 1, is in line with Linux (diverging from TCP's specification), where the $cwnd$ is given in packets. LISA is applied to the subflows if and only if they are in slow-start. LISA first finds the subflow with the largest sending rate ($old\_subflow.cwnd$, measured over the last RTT). Depending on $old\_subflow.cwnd$, between 3 and 10 packets are taken from it as *credit* and given to $new\_subflow.cwnd$. The *credit* is realized by reducing $old\_subflow.cwnd$ and hindering its increase after every ACK.

We clarify the algorithm with an example: Consider, as in Fig. 1, an existing subflow with a $cwnd$ of 40 (e.g. $old\_subflow.cwnd$=40) and a new subflow joining the connection. Since $old\_subflow.cwnd \geq 20$, 10 packets could be "taken" by the new subflow, resulting in $old\_subflow.cwnd$=30 and $new\_subflow.cwnd$=10. Then, the compound $cwnd$, whose current size is 40, should ideally become 60+20=80 after one RTT[3] (this is different from Fig. 1, where packets were already lost at this point). However, if 40 packets from $old\_subflow.cwnd$ are already in flight, the compound $cwnd$ becomes in fact 70+20=90, but here, LISA keeps $old\_subflow.cwnd$ from increasing its $cwnd$ for the first 10 ACKs. As a comparison, MPTCP would have 80+20=100 after one RTT.

LISA addresses MPTCP's *aggressiveness* described in the beginning of this section. Revisiting Figure 1(a), we observe that LISA prevents the first subflow from increasing its $cwnd$ after every ACK, hence limiting MPTCP's in flight data and reducing the number of retransmissions as shown in Figure 1(b).

[2]For simplicity, constant 1.5 KByte packet size is assumed.

[3]This assumes that the receiver ACKs every packet, which is the case with Linux in slow-start.

Algorithm 1 LISA: Called before a new subflow sends its IW

```
1:  init:  ignore_acks = false; ACKs_to_ignore = 0
2:  // get the largest sending rate among subflows in SS
3:  old_subflow = get_subflow_with_max_sendrate()
4:  if old_subflow then
5:      if old_subflow.cwnd ≥ 20  then
6:          // if old_subflow.cwnd ≥ 2*IW(10): take IW(10) packets
7:          old_subflow.cwnd –= 10
8:          new_subflow.cwnd = 10
9:          ignore_acks = true
10:     else if old_subflow.cwnd ≥ 6  then
11:         // if old_subflow.cwnd ≥ 2*IW(3): take half the packets
12:         new_subflow.cwnd = old_subflow.cwnd / 2
13:         old_subflow.cwnd –= new_subflow.cwnd
14:         ignore_acks = true
15:     else
16:         // old_subflow.cwnd < 6
17:         new_subflow.cwnd = 3 // can't take from old_subflow
18:     end if
19: else
20:     new_subflow.cwnd = 10 // no other subflow in SS
21: end if
22: if ignore_acks and inflight ≥ old_subflow.cwnd then
23:     // do not increase cwnd when ACKs arrive
24:     ACKs_to_ignore = inflight - old_subflow.cwnd
25: end if
```

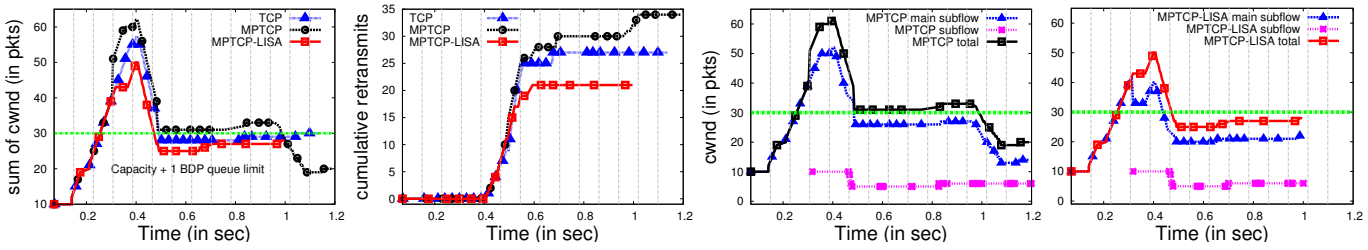
In figures 1(c) and 1(d), we illustrate the $cwnd$ of each subflow. Here, we measured 3 RTTs before the second subflow begins with its IW. This is due to MPTCP's path-manager where subflows only join the connection after the first subflow has been established. Thus, MPTCP would immediately add 10 packets to $new\_subflow$, thus amplifying the overshoot. However, LISA "takes" 10 packets from $old\_subflow.cwnd$ to $new\_subflow.cwnd$, delaying the compound $cwnd$ increase. Overall, we observe a 200 ms shorter completion time with LISA for this setup.

MPTCP's aggressiveness escalates as more subflows join the connection. Its default path-manager *fullmesh* can advertise multiple IPs in the same RTT. That means, after the first subflow has been established, new subflows *may* start sending at roughly the same time.[4] Figure 2(a) depicts the compound $cwnd$ of 3 subflows in the same scenario as shown in Figure 1. The overshoot is more significant, hence, taking longer to recover from the losses. We also observe a greater reduction in retransmissions with LISA, depicted in Figure 2(b).

IV. Measurement Setup

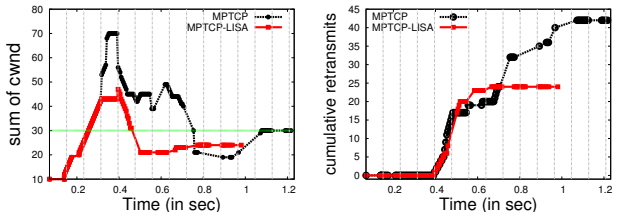
We consider two common MPTCP scenarios to evaluate LISA: a) an end-user accessing a multi-homed server shown in Section III-A and b) the datacenter scenario from [13] shown in Section III-B. The different topologies are created with the CORE network emulator [1]. Linux *netem* with *tc-htb* (default value for cburst is 1600bytes) is used to set the links' capacity and delays and *iperf* creates traffic over TCP. Data is collected at the server using *trace_printk*. All experiments are run with Linux kernel 3.14.22 and MPTCP v0.89.3.

[4]MPTCP's path-manager *ndiffports* allows new subflows to start in parallel after the first subflow is established.



(a) Total cwnd (b) Cum. retransmissions (c) MPTCP: cwnd (d) LISA: cwnd

Fig. 1. Grid lines are spaced at 70 ms, the base RTT without queuing delay



(a) Total cwnd (b) Cum. retransmissions

Fig. 2. MPTCP and LISA: 3 subflows

A. Multi-homed server

In this scenario, we limit MPTCP to 2 subflows to keep it simple and realistic. However, a larger number of subflows is possible especially when servers are connected to several ISPs or the clients are also multi-homed. Note that the 2 subflow case is in fact the worst case for LISA as illustrated in Section II: LISA plays out its benefits better with a larger number of subflows.

Here, we investigate two cases: a) the client's access link (downlink) is the bottleneck and it is therefore shared among the two flows, and b) the server's uplink connections are the bottlenecks and they are therefore not shared by the two subflows. Note that we expect to see the benefits of LISA in case a), whereas LISA may potentially be harmful in case b).

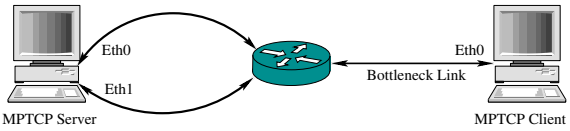


Fig. 3. Shared Bottleneck Topology

Shared Bottleneck: Here, we choose a multi-homed server with two interfaces and the client side as the bottleneck (downlink), see Figure 3. The bottleneck capacity is set to 5 Mbps, RTT is set to 40 ms and no background traffic is generated. We have selected the link speed based on the Akamai's Q1 2015 report indicating that the global average connection speed is 5 Mbps [2]. However, our results apply to different link speeds as well since the results do not depend on the raw link speeds but rather on the bandwidth×delay product (BDP) of the link. In other words, our results hold for links that have higher link speeds and shorter RTTs.

We vary the bottleneck queue size from 1 up to 34 packets (twice the path's BDP). Although a common choice for the queue size is the BDP of the link, we have also run experiments with very different queue sizes in order to stress test the algorithm under different settings. Note that multiple subflows' IW at the bottleneck could reach the path's BDP and enter CA with a small *cwnd*. We explain with an example: Assume a single flow with IW=10 traversing a path with BDP of 17 packets and a one packet queue. In the second RTT, the flow will have *cwnd*=20, i.e., two packets exceeding the path's BDP plus the queue size. Thus, the extra packets will be dropped and the flow will enter CA with *cwnd*=10, which is 55% of the available capacity! Therefore, whether the file completion time benefits from a change to MPTCP's slow-start can vary greatly with the BDP or the queue length.

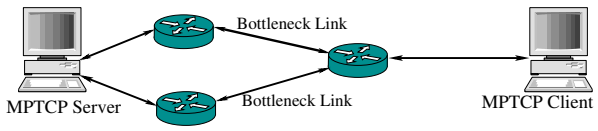


Fig. 4. Non-shared Bottleneck Topology

Non-shared bottleneck: In this scenario, there are no shared bottlenecks (see Figure 4). As before, the capacity is set to 5 Mbps and RTTs are set to 40 ms for both links.

B. Datacenter

For the datacenter scenario, illustrated in Figure 5, multiple MPTCP subflows are opened between the same pairs of end-hosts in order to evaluate the effect of Equal-Cost Multi-Path (ECMP). Because ECMP behavior is transparent to the end-hosts, some of the subflows will end up sharing the same bottleneck, while others will not – allowing LISA to play out positively in one case and potentially negatively in the other.

We want to find out whether LISA can be expected to yield an overall advantage in the use case described in [13]. While a scaled-down of the experiment is used, we observe a general trend that is expected to be valid at larger scale of the same experiment. For this, we set up a 4-ary fat tree with up to 16 nodes, RTT is set to 5ms, and the core links have a link capacity of 50Mbps. The traffic is as described in [13]: Each end-host chooses other end-host randomly, with the constraint of using only one MPTCP connection per end-host pair. Similar to [13], we randomly choose the shortest

paths for each subflow to simulate ECMP. Also, following the recommendation in [13], we set the number of subflows to 8. Here, we also vary the bottleneck queue size from 2 up to 42 packets (i.e., up to 2 BDPs).

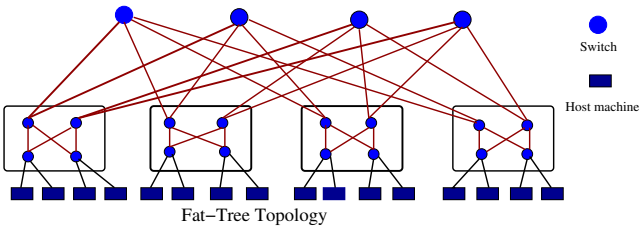


Fig. 5. Datacenter Topology

V. Performance Evaluation

In this section, we evaluated the performance of LISA for the two scenarios discussed in Section III. For each of the scenarios, we considered the following metrics in our evaluations:

- Completion time, T – the time between sending the first SYN packet to get the ACK for the last packet for a file-size transfer.
- The total number of retransmitted packets, R, and the total number of retransmitted packets before the last subflow exits slow-start, R_{ss}.
- *Sum of* $cwnd$ when the last subflow exits slow-start.

In the remainder of this section, we will take the 95% confidence interval for all the plots of the above measurement metrics.

A. Multi-homed server

Shared Bottleneck: In this scenario, we first evaluated the performance of LISA compared to MPTCP for a wide range of buffer sizes. We transferred a long enough file and only evaluated the SS phase. In Figure 6(a), we illustrate R_{ss} for varying buffer sizes. We show that LISA consistently reduces the number of retransmissions compared to MPTCP, irrespective of the buffer size, except for the very small buffer sizes. We observe that for a buffer size less than 8 packets, the first subflow leaves SS early, causing buffer overflow, and the newly established subflow sets its IW to 10 similar to the behavior of MPTCP. Hence, LISA experiences similar R_{ss} to MPTCP. Retransmissions coincide with a $cwnd$ reduction, which potentially delays transfers. In Figure 6(b), we depict the total $cwnd$ of MPTCP flows. A smaller value translates into more time to increase the window during CA, and hence, a large value is generally favorable. The figure illustrates that LISA had a larger total $cwnd$ at the end of slow-start for almost all buffer size values.

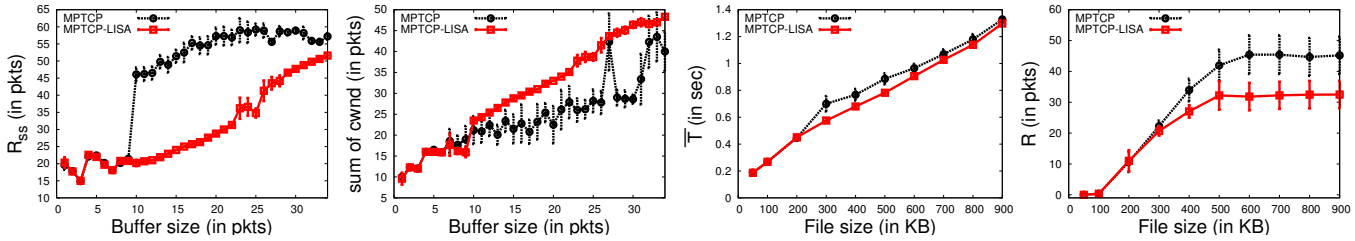
Equipped with knowledge about how buffer sizes affect the behavior of the MPTCP subflows in SS, we analyzed the completion time and retransmission performance for file transfers with sizes ranging from 50 KB to 900 KB. This range has been selected based on the HTTP transfer size statistics indicating that more than 40% of the HTTP transfers are of size up to 1000 KByte [3]. To reduce the effect of the queue size, we picked three values: 9, 17 and 30 packets. Based on Figure 6(a) and Fig. 6(b), we believe these values represent a good, medium and bad case for LISA. We then repeated the experiments 10 times for each queue size and illustrated the results in Figure 6(c) and Fig 6(d). In Figure 6(c), we observe that the completion time is improved by up to 17% for the 300 Kbyte file, and by more than 10% for files up to 500 Kbyte. For small file sizes (up to 200 KByte), note that LISA does not provide any gains since there is only one subflow (i.e. LISA is not active yet). On the other hand, for file sizes larger than 500 Kbyte, as we increase the file size, we add more and more weight to the CA phase and the performance gain decreases.

We also evaluated the benefits of LISA on the competing traffic. Intuitively, the reduced aggression of LISA is also expected to reduce MPTCP's negative impact on competing traffic. To confirm this, we ran another set of tests with a competing TCP (Reno) flow. We then compared the number of retransmitted packets of this TCP flow in Figure 7(a). The figure confirms that our intuition is indeed correct and the number of retransmitted packets experienced by the TCP flow is reduced with LISA.

So far, we have assumed that the base RTTs of the two connections are similar. If the base RTT of the connection used by the first subflow is the shortest, the number of packets given to the other subflow will be at least as much as in the equal-RTT case (10 in all our experiments). To evaluate the worst case for LISA, we were therefore only interested in the case where the first subflow sees a larger base RTT. A result from such case is depicted in Figure 7(b) where the base RTT of the first connection (used by the first subflow) is 200 ms while the other one is 20 ms; as before, the link capacity is set to 5 Mbps. We transfer a file of size 500 KB and vary the queue size from 5 to 92 packets, which is above the sum of the two BDPs (8 and 83 packets, respectively). As expected, giving packets to the shorter-RTT flow reduced the overall efficiency, but with a very small difference, as shown in the Figure 7(b).

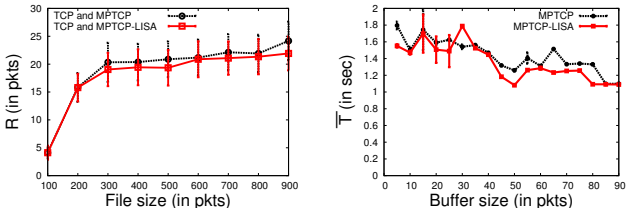
Non-shared bottleneck: Similar to shared bottleneck case, we first varied the buffer size and evaluated the R_{ss} performance of LISA for the non-shared bottleneck case in Figure 8(a). We observe that the reduced aggression of LISA lower the number of retransmitted packets by limiting the overshoot. Note that this overshoot heavily depends on the queue size, thus the benefit only plays out for a distinct set of values. The reduced aggression of LISA should at least sometimes come with a disadvantage in raw throughput. The total *cwnd* at the end of SS, depicted in Figure 8(b), shows no clear trend. The same is true for the average completion time shown in Figure 8(c) when, as before, we used the low, medium and good case queue values (from Figure 8(b), the queue sizes 9, 17 and 30 again seemed to be a good choice for these three cases).

In order to see whether the reduced number of retransmissions plays out positively, a separate test was run for a favorable buffer size of 20 packets and the results are depicted



(a) R_{ss} vs buffer size (b) Total cwnd vs buffer size (c) Mean completion time vs file size (d) Retransmissions vs file size

Fig. 6. Shared bottleneck with varying buffer size and file size



(a) Competing TCP traffic: retransmission vs file size (b) Different base RTTs: Mean completion time vs buffer size

Fig. 7. Shared bottleneck with competing traffic and different base RTTs

in Figure 8(c). As expected, we observe that the transfer time for LISA is consistently smaller. We conclude that, somewhat surprisingly, LISA does not harm transfer delay in the non-shared bottleneck case. On the contrary, LISA consistently improves the transfer delay for queue lengths where it manages to reduce the number of retransmissions.

As in the shared bottleneck scenario, we also tested heterogeneous RTT with 20 (second subflow) vs. 200 ms (first subflow) and found a small disadvantage of LISA. This disadvantage is less than 50 ms for all buffer sizes (less than a quarter of the RTT of the long subflow).

B. Datacenter

Figure 9(a) shows a consistent reduction of the number of retransmitted packets during SS, whereas Figure 9(b) shows a consistently lower $cwnd$ at the end of SS, which could play out negatively unless it is compensated for by the $cwnd$ reductions caused by the retransmissions.

As before, we determined the total transfer time per file size by running 30 tests, 10 for each good, medium and bad queue value case (10, 25 and 40). The results show that retransmits indeed had a more significant effect than $cwnd$ at the end of SS in these experiments: in Figure 9(c), the transfer completion time is significantly reduced by LISA, and this reduction correlates with the reduction of retransmits in Figure 9(d).

VI. Real-World Experiments

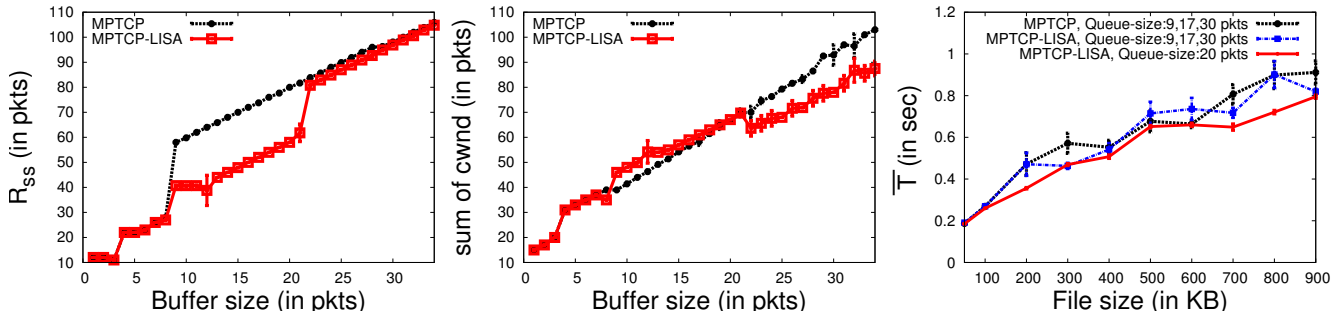
In this section, in order to validate the performance improvements of MPTCP-LISA under realistic network conditions, we conducted real-network experiments for both shared-bottleneck and non-shared bottleneck scenarios.

Setup: We constructed a multi-homed client topology, illustrated in Figure 10, using virtual machines (VM) from NorNet testbed[5] as well as four commercial cloud service providers (2x in Europe, 1x in North America and 1x in Asia). These VMs are connected via 100 Mbps links. The server machine is located in a lab network whereas the multi-homed client is a laptop connected to a wireless local network (WLAN) and a 3G mobile broadband provider. Note that 3G and WLAN have very different link characteristics in terms of capacity, packet loss and delay. Also, we build our setup to have the 3G path being the first subflow in order to revisit LISA's worst case scenario already mentioned in Section IV. Using this setup, we repeated the experiments for the non-shared bottleneck and shared bottleneck scenarios. When the wireless links are the bottleneck, this setup represents the non-shared bottleneck scenario. On the other hand, when the lab network is the bottleneck, this setup represents the shared bottleneck scenario. In order to create a bottleneck inside the lab network, we generated background traffic at the server machine towards the VM. The background traffic is composed of greedy and rate-limited TCP and UDP on/off flows. To mimic multiple applications on the same device, we further added cross-traffic on both 3G and WLAN. The device's background traffic is composed of rate-limited TCP and UDP on/off flows taking approximately 30% of the total capacity of each link.

Results: For each scenario, we collected 15 samples for each experiment. Figures 11(a) and 11(b) show MPTCP and LISA completion times in both shared and non-shared bottleneck scenarios downloading different files sizes ranging from 100 KB to 1000 KB. We observe that LISA improves the completion time for all file sizes and scenarios. For example, in the 100 KB file size case, LISA's completion time is on average reduction of 10% and 20% for shared and non-shared bottleneck scenarios, respectively. We observe that the experimental results are consistent with our emulation results. Especially, the reduction in the non-shared bottleneck scenario indicates that the queue size in our experimental setup, which is a real network setup, is a favorable for LISA.

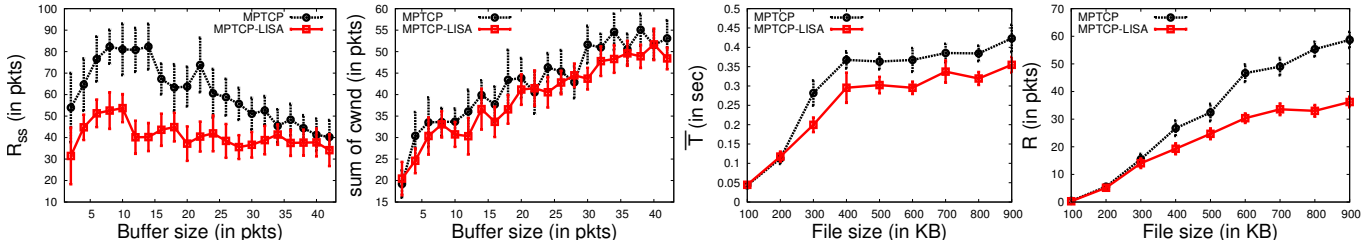
In Figures 12(a) and 12(b), we illustrate the number of retransmissions for varying file sizes. We observe that LISA reduces the number of retransmissions. One interesting observation is that LISA also limits the *bufferbloat* on the 3G path. Note that compared to MPTCP, LISA shares the 3G subflow's $cwnd$ with the WLAN subflow in slow-start. Hence, LISA avoids both subflows doubling their rate every RTT. On

[5]http://www.nntb.no



(a) R_{ss} vs buffer size (b) Total `cwnd` vs buffer size (c) Mean cmpl. time vs file size

Fig. 8. Non-Shared bottleneck



(a) R_{ss} vs buffer size (b) Total cwnd vs buffer size (c) Mean cmpl. time vs file size (d) R vs file size

Fig. 9. Data center

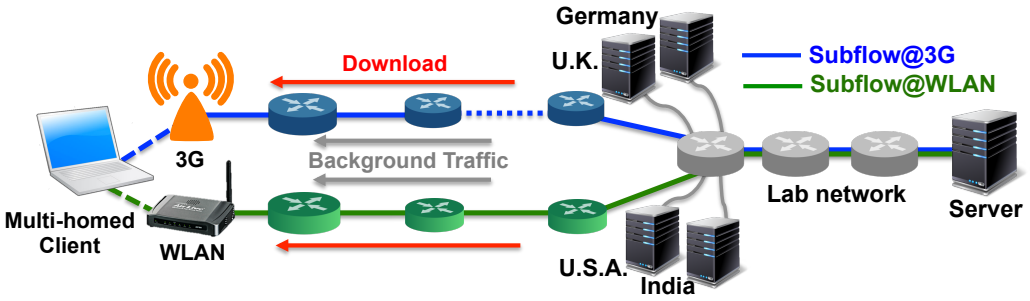
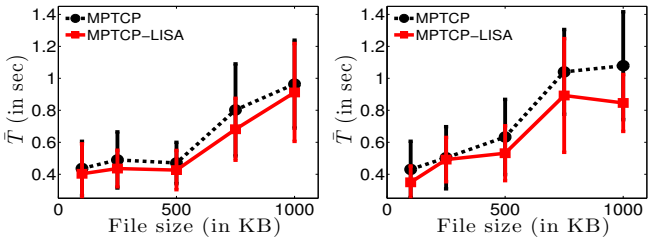
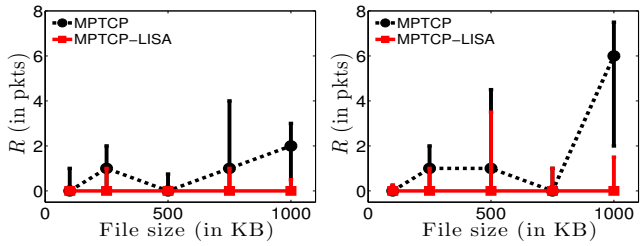


Fig. 10. Real-network experimental setup with a multi-homed client connected with 3G and WLAN to a well-provisioned lab network



(a) Shared Bottleneck (b) Non-Shared Bottleneck

Fig. 11. MPTCP and MPTCP-LISA: Completion Time



(a) Shared Bottleneck (b) Non-Shared Bottleneck

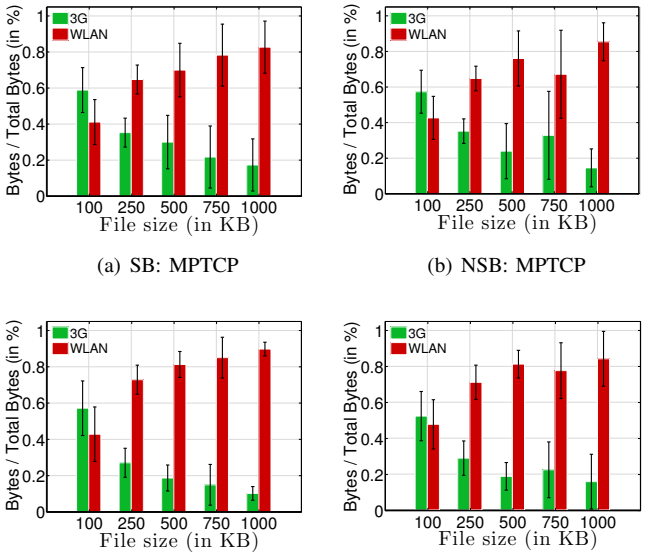
Fig. 12. MPTCP and MPTCP-LISA: Retransmissions

average, MPTCP uses 5 to 7% more of the 3G path compared to MPTCP-LISA for all file sizes in both shared and non-shared bottleneck scenarios.

Finally, we also analyzed the time for the first loss, i.e., the transition between the first slow-start to congestion avoidance phase, in all measurements. For the non-shared bottleneck scenario, LISA hit the first loss on average 50 ms later compared to MPTCP. On the other hand, for the shared bottleneck scenario, LISA's first loss occurred on average 100 ms later compared to MPTCP. Moreover, we observe less congestion avoidance occurrences in the LISA samples, suggesting that LISA finished the transfer in slow-start at times.

VII. Related Work

Several studies investigate the effect of IW10 for multiple flows. Barik et al. showed that IW10 can impact queuing delay



(a) SB: MPTCP (b) NSB: MPTCP

(c) SB: MPTCP-LISA (d) NSB: MPTCP-LISA

Fig. 13. MPTCP and MPTCP-LISA: Traffic Distribution on Each Path

and also become unfair when several flows are opened at the same time [5]. The authors in [8] showed that IW10 can create problems with multiple parallel flows, and that the impact can be reduced by SPDY (or its successor, HTTP/2.0) by reducing the number of flows. If MPTCP is used underneath SPDY or HTTP/2.0 however, it will again increase the number of flows and potentially eliminate this benefit if its subflows traverse a common bottleneck.

To the best of our knowledge, this is the first paper that investigates the current MPTCP's slow-start performance. Moreover, it provides the first attempt to couple the slow-start in MPTCP in order to overcome the unfairness of the uncoupled slow-start at the shared bottleneck.

VIII. Conclusion

In this paper, we first identify the adverse effect of MPTCP's uncoupled slow-start on the performance of MPTCP itself as well as the concurrent TCP traffic. To address this problem, we propose a link slow-start algorithm (LISA) for MPTCP that couples the MPTCP subflows during the slow-start phase. Through extensive emulations and real-network experiments, we show that LISA not only provides gains for MPTCP but also for the concurrent TCP at the bottleneck. Moreover, for the non-shared bottleneck scenario, we show that LISA does not harm, on the contrary, for favorable buffer sizes, LISA even provides gains. By coupling the slow-start, LISA effectively reduces number of retransmissions resulting in lower completion times.

While the results of our evaluation are generally quite favorable towards LISA, further improvements could be possible. For example, in the heterogeneous RTT cases that we investigated, our algorithm "took" packets from the larger RTT subflow. This could be prevented by changing the mechanism to favor subflows' $cwnd$ that have the smallest RTT. Since this can create problems related to the precision of RTT measurements and because the disadvantage was marginal (in the order of one RTT), we opted against investigating this potential improvement further at this point, but it is a possible direction for future work.

References

[1] *CORE: A real-time network emulator*, 2008. [Online]. Available: http://dx.doi.org/10.1109/milcom.2008.4753614

[2] "Akamai report on state of the internet in Q1 2015," https://www.akamai.com/stateoftheinternet, 2015.

[3] "Httparchieve," http://httparchive.org/index.php, 2015.

[4] M. Allman, S. Floyd, and C. Partridge, "Increasing TCP's Initial Window," RFC 3390 (Proposed Standard), Internet Engineering Task Force, Oct. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3390.txt

[5] R. Barik and D. Divakaran, "Evolution of TCP's initial window size," in *IEEE LCN 2013*, Oct 2013, pp. 500–508.

[6] N. Brownlee and K. Claffy, "Understanding internet traffic streams: Dragonflies and tortoises," *IEEE Communications Magazine*, vol. 40, no. 10, pp. 110–117, 2002.

[7] J. Chu, N. Dukkipati, Y. Cheng, and M. Mathis, "Increasing TCP's Initial Window," RFC 6928 (Experimental), Internet Engineering Task Force, Apr. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6928.txt

[8] Y. Elkhatib, G. Tyson, and M. Welzl, "Can SPDY really make the web faster?" in *IFIP Networking 2014*, Jun. 2014.

[9] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses," RFC 6824 (Experimental), Internet Engineering Task Force, Jan. 2013. [Online]. Available: http://www.ietf.org/rfc/rfc6824.txt

[10] R. Khalili, N. G. Gast, M. Popovi, and J.-Y. L. Boudec, "Opportunistic Linked-Increases Congestion Control Algorithm for MPTCP," IETF, Internet-draft (work in progress) draft-khalili-mptcp-congestion-control-05, Jul. 2014.

[11] C. Pearce and P. Thomas, "Multipath TCP — breaking today's networks with tomorrow's protocol," in *BlackHat USA 2014*, Aug. 2014.

[12] C. Raiciu, M. Handley, and D. Wischik, "Coupled Congestion Control for Multipath Transport Protocols," RFC 6356, Internet Engineering Task Force, Oct. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6356.txt

[13] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with Multipath TCP," in *SIGCOMM 2011*. ACM, 2011, pp. 266–277. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018467

[14] A. Walid, Q. Peng, J. Hwang, and S. H. Low, "Balanced Linked Adaptation Congestion Control Algorithm for MPTCP," IETF, work in progress, Internet-draft draft-walid-mptcp-congestion-control-01, Jul. 2014.

[15] M. Xu, Y. Cao, and E. Dong, "Delay-based Congestion Control for MPTCP," IETF, work in progress, Internet-draft draft-xu-mptcp-congestion-control-01, Jan. 2015.