Impact of Path Characteristics and Scheduling Policies on MPTCP Performance

Behnaz Arzani¹, Alexander Gurney¹, Shuotian Cheng¹, Roch Guerin² and Boon Thau Loo¹

¹University Of Pennsylvania
²University of Washington St. Louis

Abstract—With increasing deployment of Multipath TCP (MPTCP) in multihoming and datacenter scenarios, there is a need to understand how its performance is affected in practice—both by traditional factors such as RTT measurements, and by new multipath-specific considerations such as subflow selection. We carried out an initial but comprehensive study using an actual MPTCP implementation in an emulated network environment, to explore the impact of different factors on MPTCP throughput. We find that path selection and packet scheduling have a large effect on performance, and that merely trusting the congestion control mechanism to do the right thing is not enough. Moreover, we provide evidence that throughput can be improved by slight modifications to the send buffers and path selection components of the implementation. Important challenges in network design remain, if only to ensure that multiple suitable paths exist in a network.

1 INTRODUCTION

Multipath TCP (MPTCP) is one proposed solution for many networking scenarios where a given data flow might exceed the capacity of a single path, but where the total capacity of the network is sufficient for the total amount of data being transmitted. MPTCP [2], presents itself to the application layer as a conventional TCP connection, but actually multiplexes the traffic across a bundle of parallel paths, with joint congestion control. In this way, traffic volume can expand and contract as required on each sub-flow, as their relative characteristics (congestion, loss rate, etc.) change over time. Possible application areas range from datacenter deployments—where topologies are highly structured and reliable—to the wider Internet, where multihoming may yield paths that are heterogeneous and volatile.

In this paper, we carry out experiments to observe how actual implementations perform in heterogeneous settings. Our ultimate aim is to use these data to understand the role of path selection in MPTCP performance. Our initial results show that the MPTCP congestion control mechanisms do not obviate the need to manage the selection of sub-flows, and that the choice of packet scheduler has a large impact on throughput. We also demonstrate that tuning a network for TCP does not mean that peak MPTCP performance will be achieved. In particular, simply using any number of paths with the best TCP performance as the MPTCP sub-flows is not necessarily the optimal choice. We evaluated the possible impact of various types of schedulers on MPTCP throughput. Our results indicate that using an optimized scheduler, tailored specifically for MPTCP’s use, will have significant benefits in improving throughput. Modification of the send buffers is also helpful in this regard.

2 BACKGROUND ON MPTCP

Figure 1 gives an overview of the fundamental structure of MPTCP. The design is motivated by the overriding need to be compatible with applications and network devices which are unaware of MPTCP. Therefore, upper layers of the protocol stack only need to deal with a logical ‘master’ TCP socket; beneath this are several subflows, each of which is a conventional TCP connection, so that middleboxes which know how to deal with TCP traffic do not need to be changed in order to handle MPTCP.

The advantage of this structure, compared to an application which creates multiple TCP connections on its own, is that the MPTCP layer is able to take care of out-of-order packet arrivals on the sub-flows, and handle congestion control among them. Indeed, without this, a sender would be able to seize more than its fair share of network resources [2], and conventional congestion control is known to be inadequate in achieving high throughput when used over multiple paths. The reason is that packets along multiple paths frequently arrive out of order at the receiver, resulting in duplicate ACKs. The mere TCP sender misinterprets this as a sign of congestion, and upon receiving the third duplicate ACK, reduces the send window size, hence penalizing throughput.

MPTCP’s congestion control algorithm was designed with the following objectives:

- It should perform at least as well as a single TCP session on the best path.
- It should be fair to other TCP flows.
- It should move traffic away from congested paths to less congested ones when possible.
To satisfy these goals, MPTCP performs joint congestion control over its multiple paths. This joint congestion control operates exactly as TCP would when in slow start or fast retransmit. However, MPTCP couples the tuning of the congestion windows of each sub-flow during congestion avoidance.

A significant body of work has been devoted to characterization and optimization of MPTCPs coupled congestion control. The authors of [7] perform a theoretical analysis of the problem, characterizing factors governing responsiveness, TCP-friendliness, and window oscillations of a joint congestion control method for MPTCP. Other work such as [11], [3], and [14] are further examples of work focused on improving MPTCP joint congestion control.

The method of [14] is the basis of our work in this paper. We do not claim that it is optimal (indeed, it is not [3]), but it is simpler and more intuitive, and for the scenarios we explore, it coincides with the improved method of [3]. During congestion avoidance, this Linked Increase Algorithm (LIA) regulates traffic by adjusting the sending window on each sub-flow. Assume path \( i \) has a round-trip time (RTT) of \( r_i \), and a TCP congestion window of \( w_i \), where for each such \( w_i \), \( \hat{w_i} \) represents the value at equilibrium. In this case, during congestion avoidance MPTCP will:

- Upon reception of each ACK on sub-flow \( i \), increase the window on that sub-flow by \( \min \left( \frac{\alpha}{w_{\text{tot}}}, \frac{1}{w_i} \right) \), where \( w_{\text{tot}} \) denotes the total congestion window across all sub-flows.
- Upon each loss on sub-flow \( i \), reduce the window on that sub-flow by \( \frac{1}{w_i} \).

The parameter \( \alpha \) in the above formulations is chosen as

\[
\alpha = \hat{w}_{\text{total}} \left( \frac{\hat{w}_i}{r_i} \right)^2, \tag{1}
\]

in order to satisfy the design goals of MPTCP and compensate for the effects of different RTTs. We will further discuss these effects in Section 3.

We have not yet described how the sub-flows are made available to the MPTCP implementation. The current implementation of MPTCP [9] attempts to add all possible endpoint addresses (and therefore paths), and incorporates an API to add and remove select addresses from the available set. Note that there is no mechanism, either on the application or the provider side, for supplying paths with particular characteristics (such as a target RTT), or for ensuring disjointness. Once paths are established, the only way to discover a path’s quality is to use it and find out. This ignorance of path quality may have a significant impact on achieved throughput.

3 Emulation Based Study of MPTCP

In this paper, through an emulation based study, we aim to explore the following challenges associated with MPTCP, in order to shed some light on its performance characteristics. We focus on three aspects: path characteristics, configuration parameters, and scheduling policies. Combined with previous work on MPTCP in data centers [10], [8], OpenFlow [13], and PlanetLab [4] our work can help with providing a more comprehensive understanding of MPTCP performance.

Path Characteristics. In order to devise strategies for improving MPTCP throughput, we must understand the influence of path characteristics on its behavior. To our knowledge, a comprehensive evaluation of its performance under varying path characteristics does not yet exist, and hence we lack the necessary data to design an application-level path selection mechanism, or its provider-side complement whereby paths are provisioned.

The authors of [1] describe practical concerns when implementing a deployable MPTCP in the Linux kernel. More importantly, [1] points out that using a path with high RTT may result in the receive buffer size growing beyond the allowed maximum. Other work such as [9], and [14] provide further evidence of this problem. In addition, [1], [9], and [14] all provide experimental evidence as to how MPTCP performance suffers when the size of the receive buffer is limited. However, all of these works focus on example configurations and do not focus on identifying how the penalty changes under different path characteristics.

The concern with using paths of different RTTs is not limited to the effect on the size of the receive buffer. Although MPTCP’s joint congestion control makes sure that the paths of higher RTT are used less often than those with lower RTT, it maintains a minimum rate of transmission on these paths in order to probe for any changes in path characteristics. This behavior may greatly impact MPTCP performance as the application would have to wait for the missing segments before it can read from the receive buffer.

The authors of [9] have suggested various fixes to this problem. They propose to opportunistically retransmit packets sent on slower paths, on a path with a shorter RTT whenever the slower path causes the receive buffer to block transmission. In such circumstances, [9], also proposes to penalize slow sub-flows every RTT by shrinking their respective send windows by a factor of two and modifying their ssthresh if congestion has already been detected on that flow. Though these fixes help

1. Since the Linux kernel starts with ssthresh equal to \( \text{TCP\_INFINIT\_SSTHRESH} \), it suffices to check for \( \text{ssthresh} \neq \text{TCP\_INFINIT\_SSTHRESH} \).
in mitigating the impact of different RTTs, they do not eliminate it, nor the need to quantify its magnitude.

Configuration parameters. One of the goals of our study is to explore the impact of configuration parameters on the performance of MPTCP, particularly under different path selection policies. During our experiments we also noticed that the size of the send buffer can significantly impact the extent to which MPTCP can utilize its paths. Given that MPTCP’s congestion control is responsible for ensuring fairness, an MPTCP user may selfishly tune its send buffer without worrying about its impact on other users’ throughput. Assuming that sufficient memory exists, our experiments show that there is merit in increasing the size of the send buffer to allow more efficient utilization of the paths by MPTCP.

MPTCP scheduler. Finally, the master socket of MPTCP is in charge of striping packets across the multiple subflows. The master socket’s scheduler is the entity in charge of making the decision of which subflow the next segment should be sent on. Providing a preliminary understanding of the impact of different scheduling policies on MPTCP throughput, and on the impact of mismatched RTTs, is the third and final main objective of this paper.

4 Experiment Setup

We use the Linux kernel implementation for MPTCP described in [9], executed using NS3’s direct code execution software (ns3-dce) [5], [12]. This setup allows us to experiment with extreme scenarios in a controlled environment. We used Ubuntu 13.04 with a 32-bit Linux kernel at version 3.8.0. Note that the smoothed RTT estimator [6] implementation is not specifically tailored for a 64-bit kernel; as described below, the built in MPTCP scheduler is highly sensitive to this estimate and the result is a significant difference in path utilization. We therefore recommend the 32-bit implementation for NS3 until the situation is resolved.

We set up the simple topology shown in Figure 2. Here, R1, R2, and R3 represent Internet routers, and S and D are the source and destination respectively. The link between R3 and D forces MPTCP to establish only two subflows: since otherwise both S and D would have two interfaces, there would normally be four different possible combinations, and so two subflows would share each path. The link is given an artificially high bandwidth of 1 Tbps, and zero latency, so as not to affect bottleneck bandwidth nor total latency; The total round trip time ri on each path i is split equally between the path’s two links, so each link has a one-way delay of ri/4. Each simulation run, lasting 500 seconds of simulated time, uses iperf to send traffic and measure the goodput, characterized as the total bytes received at the destination.

Including all headers, packets are sized at 1502 bytes. The links have been configured to have no losses, and unless noted otherwise all data points have been collected using a send buffer of 64 KB, and a maximum window of 34 segments (1460 bytes each) on each subflow.

MPTCP includes a scheduler component to assign packets to subflows. To gauge the effect of the scheduler design on performance, we repeated our experiments using the following four schedulers.

1) The RTT-Aware Scheduler The Linux kernel MPTCP implementation includes a scheduler that gives higher priority to paths with lower RTT [9]. When a segment is ready for transmission, the assigned path will be the one of minimum RTT, out of all those subflows whose (sending) congestion window is not yet full. If there is more than one such path, then the scheduler makes a choice with a systematic preference towards one of the paths, and continues to favor this path until its congestion window becomes full; it will again return to this path once space is available. The scheduler therefore has a ‘sticky’ nature, with implications for throughput discussed in Section 5.

2) RTT-Aware Scheduler With Random Tie Break A modification to this scheme, trying to eliminate the ‘sticky’ effect, uses a random tie breaker in all cases of equal RTT.

3) The Round Robin Scheduler Alternatively, paths may be chosen in a round-robin fashion out of those whose congestion window is not yet full.

4) The Random Scheduler Finally, packets may be assigned to subflows at random, regardless of RTT.

5 Evaluation

We now present results on the impact of path characteristics on MPTCP throughput, the role of the scheduler on performance in the congestion avoidance stage, and the impact of the size of the send buffer. Aside from the experiments in Section 5.2, all results shown here use the RTT-aware scheduler. All optimizations described in Section 3 have been enabled; while our experiments appear to confirm that these changes improve performance, we do not carry out a detailed analysis here.

5.1 Path Characteristics

In our first set of experiments, we explore the impact of path characteristics on the overall performance of MPTCP. Since the main use of MPTCP is to allow better resource pooling, it is important to understand the situations in
which this transport layer protocol would be able to provide the most benefit.

Section 3 explained how RTT diversity may result in a significant decrease in MPTCP throughput. The magnitude of the effect could depend on many factors: here, we analyze the role of bandwidth and of the difference in RTT across all paths. Other possible factors, including temporal variation of RTT, are deferred to future work.

In order to explore the impact of RTT difference, in each experiment, path $a$ is set to have the lower RTT (denoted $r_a$), and path $b$ an RTT of $r_b = r_a + \delta$ for some $\delta \geq 0$. The paths will be of some equal bandwidth $C$.

Figures 3a, 3b and 3c depict iperf goodput for $r_a \in \{100,200,300\}$ ms and $0 \leq \delta \leq 80\text{ms}$, with $C \in \{0.5,1,5,10\}$ Mbps. The goodput has been normalized to the goodput of a single-path user of path $a$.

Note, that $C \geq 5$ Mbps exceeds the maximum possible throughput\(^2\) achievable by either TCP or MPTCP, given $r_a$. This implies that in such situations, MPTCP may achieve the maximum possible throughput by using only its best path. The chosen set of bandwidths, therefore, spans two possible regimes: one where the bandwidth of any single path is greater than the maximum throughput that either TCP or MPTCP can achieve, and one where individual path bandwidths are all below this value.

Our results in Figures 3 and 4 confirm that the use of MPTCP is of little benefit when $C \geq 5$ Mbps. Indeed, as MPTCP maintains a minimal use of $b$, its throughput suffers as $\delta$ is increased, making single-path TCP a better choice under such circumstances, and this in spite of the use of optimizations such as those suggested in [9].

We also looked at two possible scenarios for the initiation of an MPTCP connection. Figure 3 shows the results when the source $S$ initiates the connection using path $b$, after which it informs $D$ of its additional interfaces and adds the path $a$. Figure 4 shows the results when $a$ is used to initiate the connection. The results show, quite surprisingly, that the order in which paths are established has a significant impact on MPTCP performance. It is indeed true that starting with the wrong path should penalize performance, since it will take the better path longer to grow its window to the maximum possible value. However, one would expect that the impact of an original ‘mistake’ in choosing paths would decay over time resulting a much smaller relative decrease in performance in the overall goodput. We have not yet been able to localize the cause of this behavior, and thus leave it for future work.

Our results confirm the intuition that differences in path characteristics impact MPTCP performance. The next section examines whether a scheduler can help reduce the protocol’s sensitivity to these differences.

### 5.2 Scheduling Policies

The next set of experiments seeks to assess the extent to which the use of different schedulers affects MPTCP’s performance. In particular, whether it can help cope with paths of different characteristics? A sub-flow’s sending window already provides by itself a means to control its rate and favors the use of paths with lower RTT. A scheduler shares those goals, and its path selection decision may, therefore, interact positively or negatively with the window control mechanism. At the very least, the effect of those interactions is unclear, and exploring is what motivated our experiments with different schedulers.

Figure 5a shows iperf’s normalized goodput for four different schedulers while varying $\delta$. We focus on a scenario where MPTCP can improve throughput, namely, $C = 1$ Mbps. The RTT of path $a$ is set to $r_a = 200\text{ms}\(^3\)$. Intuitively the RTT-Aware scheduler should be able to minimize the use of the slower path, which minimizes the number of packets the application would need to wait for before reading from the receive buffer. This property should allow the scheduler to improve performance when paths have unequal RTTs. However, our results appear to indicate that RTT awareness in scheduling can in some sense amplify path heterogeneity, and offers therefore limited benefits unless paths heterogeneity is substantial. In contrast, although Round Robin and Random schedulers are oblivious to RTT differences, their more evenly distributed transmission decisions make for a steadier transmission rate while benefitting from the fact that the window control mechanism is already accounting for RTT differences.

The results of Figure 5a notwithstanding, RTT-Aware schedulers have some benefits. Figure 5b and 5c show throughput results when the bandwidth $C_b$ of path $b$ is varied, for RTT values of $r_b = 100\text{ms}$ and $140\text{ms}$, respectively. The goal is to illustrate how effectively schedulers utilize additional path $b$ capacity. The results show that among all four schedulers, the RTT-Aware scheduler makes the most of path $b$ additional bandwidth. This persists even as the RTT of path $b$ increases. The likely explanation is that because it exhausts the window on one path before moving on to the next, the RTT-Aware scheduler is able to minimize its idle time\(^4\).

Our experiments clearly show that further analysis is needed to characterize the best scheduler for an MPTCP implementation. Our results further indicate that it is highly likely that the optimal scheduling strategy depends on the characteristics of the paths being used.

### 5.3 Send Buffer Size

Our analysis of the run time logs of the experiments in Sections 5.1 and 5.2 showed that the choice of a 64 KB send buffer reduced MPTCP’s overall performance gain. In this section we illustrate the impact of the send buffer size on iperf goodput.

Figures 6a, and 6b show iperf goodput under two different send buffer settings and for $C = 10$ Mbps and $C = 5$ Mbps.

2. For a TCP user, this is given by $\min(s,w)$, where $s$ denotes the size of the send buffer and $w$ the maximum window size.

3. Experiments for $r_a = 100$, and $r_a = 300$ produced similar results.

4. All experiments in this section were repeated with the optimizations described in [9] turned off. The results remain roughly the same, indicating that the impact of the schedulers is independent of the optimizations and may be considered separately.
Fig. 3: Impact on iPerf throughput when increasing the difference in RTT’s of the two paths. Here, the source, $S$, initiates the connection using path $b$.

Fig. 4: Impact on iPerf throughput when increasing the difference in RTT’s of the two paths. Here, the source, $S$, initiates the connection using path $a$.

Fig. 5: Impact on iPerf throughput when using different schedulers, 5a shows their performance while varying $\delta$. 5b, and 5c depict the performance while varying $C_b$.

1Mbps, respectively. Both figures indicate that increasing the size of the send buffer one improves MPTCP’s throughput. This is not surprising, as using a higher send buffer with a normal TCP connection will also improve throughput. However, Figure 6c, which depicts the change in iPerf’s normalized goodput\(^5\), shows that one derives a much higher benefit with MPTCP from increasing the size of the send buffer (obviously assuming the system has enough memory at its disposal).

As a final note, we can see that both Figures 6a, and 6b indicate that part of the RTT-Aware scheduler’s success in minimizing use of the higher RTT path (for large enough $\delta$ values) is due to the a smaller send buffer. Although the RTT-Aware scheduler’s throughput is higher with a larger send buffer, a small send buffer enforces an implicit bound on the maximum amount of traffic that it allocates to path $b$: reducing its sensitivity to differences in path characteristics. Thus, there exists a trade-off in increasing the size of the send buffer when using the RTT-Aware scheduler.

5. In each case the goodput has been normalized to the single path TCP goodput using the same send buffer size.


6 Summary and Ongoing Work

Our paper provides a preliminary study on the impact of path characteristics and scheduling policies on the performance of MPTCP. We briefly summarize our results as follows:

- Depending on path characteristics (RTT and bandwidth), there may be situations where both TCP and MPTCP result in roughly the same overall performance, in which case using vanilla TCP would be better given the unnecessary overhead introduced by MPTCP. Interestingly, we observe that MPTCP could actually perform worse than TCP under some circumstances, furthermore, the order in which paths are selected have a significant impact on its performance.

- MPTCP’s coupled congestion control is in charge of controlling the split of traffic across available sub-flows. However, it does not come into play unless the protocol has moved to the congestion avoidance stage. We further observe that the choice of the best scheduling policy also depends on path characteristics.

- The send buffer size has significant impact on MPTCP’s performance. Our experiments support the idea that a larger send buffer could allow MPTCP to provide higher performance gains.

Through this preliminary study, we have identified several interesting questions that we are currently exploring.

Among these questions: why does the choice of the initial sub-flow impact MPTCP throughput to the extent that it does, and why is the protocol not able to make up for its initial mistake? What type of scheduler would maximize MPTCP’s performance benefits? Can ISP’s optimize MPTCP performance by modifying their multihoming strategies? If so, how would this change impact normal TCP performance? How hard can it be? designing and implementing a deployable multipath tcp. In ACM SIGCOMM Computer Communication Review, volume 41, pages 266–277. ACM, 2011.

7 Acknowledgements

This research has received funding from NSF grants: CNS-1218066, CNS-0845552, ITR-1138996, CNS-0915982, and as well as AFOSR Young Investigator Award FA9550-12-1-0327. We would also like to thank our anonymous reviewers for their constructive feedback.

References


Fig. 6: Impact of changing the size of the send buffer.

6. Note, that this does not occur if no packets have ever been dropped as the initial asthresh is too high.