MMPTCP: A Multipath Transport Protocol for Data Centers

Morteza Kheirkhah, Ian Wakeman, George Parisis
School of Engineering and Informatics
University of Sussex, UK
Email: {m.kheirkhah, ianw, g.parisis} @sussex.ac.uk

***Abstract*—Modern data centres provide large aggregate network capacity and multiple paths among servers. Traffic is very diverse; most of the data is produced by long, bandwidth hungry flows but the large majority of flows, which commonly come with strict deadlines regarding their completion time, are short. It has been shown that TCP is not efficient for any of these types of traffic in modern data centres. More recent protocols such MultiPath TCP (MPTCP) are very efficient for long flows, but are ill-suited for short flows.**

In this paper, we present Maximum MultiPath TCP (MMPTCP), a novel transport protocol which, compared to TCP and MPTCP, reduces short flows' completion times, while providing excellent goodput to long flows. To do so, MMPTCP runs in two phases; initially, it randomly scatters packets in the network under a single congestion window exploiting all available paths. This is beneficial to latency-sensitive flows. After a specific amount of data is sent, MMPTCP switches to a regular MultiPath TCP mode. MMPTCP is incrementally deployable in existing data centres as it does not require any modifications outside the transport layer and behaves well when competing with legacy TCP and MPTCP flows. Our extensive experimental evaluation in simulated FatTree topologies shows that all design objectives for MMPTCP are met.

I. Introduction

Modern data centre network architectures [1]–[3] provide very high aggregate bandwidth and dense interconnectivity in the network by incorporating multiple paths among servers. They support a large number of network services which produce very diverse intra-data centre traffic matrices. The majority of the data is produced by long flows, which are bandwidth-hungry. Short flows commonly come with strict deadlines regarding their completion time. According to [2], "99% of flows are smaller than 100 MB, however, more than 90% of bytes are in flows between 100 MB and 1 GB". If short flows cannot deliver all their data before their deadlines, some results may be discarded, decreasing the overall quality of the main computation or forcing some tasks to be restarted, wasting CPU and network resources. Deadlines are typically missed due to encountering transient and/or persistent congestion in their paths. Short flows result in very bursty and unpredictable traffic patterns, which in turn means that data centres are susceptible to severe transient congestion in any link in the network.

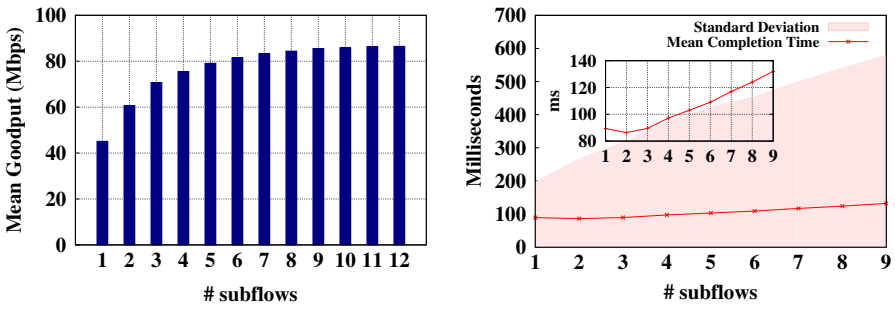
To utilise the available multiple paths through the network, *Equal-Cost Multi-Path* (ECMP) routing [4] is deployed to route flows across the multiple paths. However, even with ECMP in place, TCP is ill-suited for both long and short flows within the data centre. Under high load, long flows collide with high probability and, as a result, network utilisation significantly drops and only 10% of the flows achieve their expected throughput [5]. TCP is also inefficient for short flows, especially when competing with long flows. Queue build-ups, buffer pressure and TCP Incast combined with the shared-memory nature of data centre switches results in short TCP flows often missing their deadlines mainly due to retransmission timeouts (RTOs) [6].

Several transport protocols have been recently proposed to deal with these challenges. DCTCP [6], D2TCP, [7] and D3 [8] all aim at reducing flow completion times for latency-sensitive flows. However, they require modifications in the network and/or deadline-awareness at the application layer. Such information may not be known a priori (i.e. at connection time). Worse, these protocols are not designed to co-exist with other transport protocols, and thus have a problematic deployment path.

Multipath transport protocols, such as MultiPath TCP (MPTCP) [9], transfer data using multiple subflows and rely on ECMP to distribute the subflows to several network paths. As shown in [5], MPTCP achieves high goodput and improves the overall network utilisation. This is also illustrated in Figure 1(a)[1], where MPTCP with eight subflows almost doubles the application goodput when compared with TCP (i.e. MPTCP with a single subflow in Figure 1(a)). However, MPTCP handles short flows inefficiently. The congestion window of a subflow may be very small over its lifetime. As a result, even a single lost packet can force an entire connection to wait for an RTO to be triggered because this lost packet cannot be recovered through fast retransmission. This is clearly illustrated in Figure 1(b), where the mean short flow completion time increases as more subflows are used (better shown in the embedded Figure). Note that the number of connections that experience one or more RTOs significantly increases as well, hence the increase in the standard deviation. Even a single RTO may result in flow deadline violation.

Central flow scheduling approaches, such as Hedera [11], only deal with long flows. Hedera detects long TCP flows at the edge switches and its central controller schedules these to optimise bandwidth allocation. Short flows are not

[1]For these simulations we used our custom implementation of MPTCP in ns-3 [10].



(a) Goodput

(b) Flow Completion Time

Fig. 1. Goodput for long flows (1(a)) and completion time for short flows (1(b)) in a full-bisection and 4:1 over-subscribed FatTree topology consisting of 128 and 512 servers, respectively. One third of the servers run long (background) flows. The rest run short flows (70KBs each) which are scheduled according to a Poisson process). All flows are scheduled based on a permutation traffic matrix.

considered at all, therefore their completion times suffer from the aforementioned TCP pathologies.

Supporting and running multiple transport protocols in a data centre can be problematic. Fairness among different protocols is difficult to achieve; most protocols for latency-sensitive flows are not compatible with TCP or MPTCP [6], [7]. Running multiple transport protocols is also a burden for application developers who would have to decide upon the most suitable transport protocol. Both application requirements and data centre topologies evolve over time and so a transport protocol that performs well over disparate topologies and traffic matrices is a necessity.

In this paper, we present MMPTCP, a multipath transport protocol that aims at:

1) high throughput for large flows;
2) low latency for short flows;
3) tolerance to sudden and high bursts of traffic;
4) minimal changes to the network architecture
5) fair co-existence with other transport protocols.

MMPTCP achieves its objectives by transporting data in two phases. Initially, it randomly scatters packets in the network under a single congestion window exploiting all available paths. This is beneficial to latency-sensitive flows, which typically have bursty traffic patterns. After a specific amount of data is sent, MMPTCP switches to a regular MultiPath TCP mode, efficiently handling long flows through separate congestion windows for each subflow.

The remainder of this paper is as follows: in section II, we present the design of the proposed transport protocol and its influences from Packet Scatter [12] and MPTCP [5]. We also describe the problems associated with scattering packets in the network and packet reordering, when multiple paths are used, and discuss our proposed solution. Section III presents our extensive evaluation of MMPTCP in simulated data centre topologies. Simulations are based on our MMPTCP implementation in Network Simulator-3 (ns-3). Section IV explores potential future improvements of MMPTCP with respect to the congestion control algorithm used during the first phase of our protocol, multi-homed data centre topologies and QoS features that are available in modern data centres.

II. Design

In this section, we briefly discuss Packet Scatter (PS) and MPTCP protocols before describing MMPTCP, which has been designed based on these two protocols. We also discuss spurious retransmissions due to packet reordering, which are a key challenge for MMPTCP, and describe our solution which is embedded in the proposed protocol.

A. Packet Scatter

Data transport through scattering (spraying) packets in the data centre has been briefly explored in [5] and discussed in more details in [12]. The key idea behind Packet Scatter (PS) is that ECMP network switches choose one of the valid output ports on a per-packet instead of on a per-flow basis, as in Valiant Load Balancing [13]. Traffic can thus be distributed as evenly as possible among all paths between two endpoints. The corollary of packets within a flow taking multiple paths is that packet reordering becomes more likely and so the protocol must use more robust Fast Retransmit algorithms to deal with out-of-order packets.

It has been argued that if traffic load is equal among servers and a data centre has a uniform network topology, such as FatTree [1] or VL2 [2], then PS achieves perfect load balancing in the network core and eliminates congestion from that layer [5]. However, although traffic that is switched on a per-packet basis does not create hotspots at the network core, traffic that is distributed through ECMP on a per-flow basis (e.g. TCP or MPTCP flows) may still end up sharing links and causing congestion. Network hardware failures or traffic, which typically consists of regular TCP flows flowing from the Internet to the data centres, may also cause such congestion [5]. Since PS flows share a single congestion window, if a packet is dropped, then the congestion window shrinks across all paths that the PS flow is using, drastically reducing throughput. The longer the flow, the more likely that the flow will encounter congestion, and so PS can have a diminished performance for long lasting flows.

B. MultiPath TCP

MultiPath TCP is an extension of TCP that transfers data through multiple paths simultaneously. It actively senses net-

work congestion for all its subflows and shifts traffic from more to less congested paths. Unlike TCP, MPTCP deals with network congestion gracefully by putting fewer packets to the congested subflows. The main requirement to achieve such behaviour is to retain a congestion window for each subflow and link each of them together. This is the main reason why MPTCP (with at least eight subflows) doubles the mean goodput of long flows, compared to regular TCP, as illustrated in Figure 1(a).

MPTCP reacts to congestion very quickly. It can remove traffic from congested links within a few RTTs (unlike Hedera [11]), therefore dealing efficiently with the traffic concentration problem. MPTCP is an appealing approach for data centres characterised by an extremely dynamic nature. However, because it has multiple congestion windows, it is very susceptible to timeouts when a flow only contains a few packets. Packet drops from each congestion window may cause an entire MPTCP connection to hold for a retransmit timer to be triggered. As illustrated in Figure I, increasing the number of subflows is beneficial to the goodput of long flows, but it is harmful to flow completion of short flows.

The question that is raised is whether it is possible to adjust the number of MPTCP subflows based on the size of the flow. It has been argued that some applications can provide high-level information, such as flow size [8], to the transport layer. If such information was available, one could decide how many subflows it might be effective to use. For example, in the case of short flows, it is better to have a single subflow. Unfortunately, the majority of applications do not expose their flow sizes to the end-hosts, (i.e. the network stack is unaware of this high-level application information). MPTCP cannot have any indication about how many subflows to open for a flow; if a predefined number of subflows is used for all types of flows then MPTCP is likely to significantly damage the flow completion time of short flows.

C. MMPTCP: Combining PS with MPTCP

Before delving into the mechanics of MMPTCP, we briefly enumerate its main design principles and objectives:

1) Enforcing handling of short flows through scattering packets in the network, preventing MPTCP's short flow inefficiencies.
2) Enforcing handling of long flows through standard MPTCP.
3) Decreasing the burstiness of data centre networks, which mainly originates from short flows, by diffusing packets throughout the network effectively preventing transient congestion in the network core.

The core idea behind MMPTCP is that, initially, data is transferred by scattering packets in the network until the amount of transmitted data reaches a certain threshold. To do so, we employ source port randomisation at the source host and standard ECMP at network switches. A token[2] is added to each packet of the initial subflow as a connection identifier so that a randomised packet can be forwarded to the corresponding MMPTCP connection correctly. Note that the standard connection identification through the 5-tuple is no longer valid during the initial phase of data transmission because of the source port randomisation. Data transport is governed by a single congestion window throughout the duration of the first phase, whose aim is to take advantage of all available network paths and quickly complete short flows.

When the switching threshold is reached, MMPTCP switches to standard MPTCP with multiple subflows to benefit from MPTCP's efficiency in dealing with long flows. The initial subflow is only allowed to scatter packets in the network during the first phase; after switching to MPTCP, no more packets are put in the initial subflow, which is deactivated (but not closed[3]) when its window gets emptied. To ensure continuity of data transmission, after the switching threshold is reached, the initial subflow becomes deactivated only when at least one new MPTCP subflow is established. In other words, after switching to MPTCP no more data is placed on the initial subflow, which is ignored by the MPTCP congestion controller. During the second phase, data transmission is governed by MPTCP's congestion control mechanism.

In the initial handshake of MPTCP, SACK may also be activated if DSACK is used as a part of the packet reordering strategy (see Section II-D). MPTCP works with SACK, so there is no problem in having SACK activated over the lifetime of an MMPTCP connection. On the other hand, DSACK would only be used in the initial phase to detect and mitigate spurious retransmissions due to out-of-order packets.

D. MMPTCP and Packet Reordering

A TCP sender receives a duplicate acknowledgement (duplicate ACK) when a packet gets dropped, delayed or reordered. It enters the Fast Retransmit phase upon the arrival of the third duplicate ACK for a missing packet (when the duplicate ACK threshold parameter is set to three). It retransmits the perceived lost packet and halves its congestion window as a reaction to the congestion signal. However, the Fast Retransmit mechanism may still be falsely triggered when a reordered packet reaches the receiver after it has sent a third duplicate ACK. This condition may lead to spurious retransmissions of reordered packets even if no loss has occurred. In other words, the sender interprets the reordered packet as lost. As a result, the sender falsely triggers the Fast Retransmit mechanism and halves its congestion window, which, in turn, leads to performance degradation.

Although this condition is very unlikely to occur with TCP/MPTCP flows in data centres, it is common when scattering packets in the network, since RTTs on different network paths may vary over time due to queuing delays. MMPTCP must therefore handle packet reordering during its first phase in order to be able to meet its objectives with respect to completion times of short flows.

[2]A token is a locally unique identifier assigned to a MMPTCP connection upon establishment.

[3]The initial subflow is the only subflow presented to the application and if it was closed, the connection would lose its identity.

Setting the right *dupthresh* value is not trivial; if *dupthresh* is too low, spurious retransmissions become the norm. If it is too high, the sender may react to congestion through a retransmission timeout instead of the Fast Retransmit mechanism; obviously this would be a very undesirable situation as even a single timeout may lead a flow to miss its deadline. Our experimental evaluation in Section III-D confirms these observations.

There are three key aspects in making TCP more robust to packet-reordering: preventing, detecting and mitigating spurious retransmissions due to out-of-order packets.

One well-known solution for detecting and mitigating spurious retransmissions is DSACK [14], which is an extension of SACK TCP [15]. SACK TCP can deal with multiple packet drops much faster than other versions of TCP (e.g. NewReno [16]). This is particularly beneficial for latency-sensitive flows. When a spurious retransmission is detected by DSACK, the state of the congestion window can be simply reversed to the state when a loss is detected.

One possible approach for preventing spurious retransmissions is to dynamically adjust the *dupthresh* parameter based on information that can be retrieved from DSACK, SACKs, ACKs, RTOs and Fast Retransmits. RR-TCP [17] follows a similar approach. RR-TCP attempts to adjust the *dupthresh* value dynamically by understanding the maximum distance in packets by which a segments is displaced, based on feedback from the network.

Our novel approach for preventing out-of-order packets is to set the value of *dupthresh* based on topology-specific information. For example, FatTree's IP addressing scheme can be exploited to calculate the number of available paths between a sender and a receiver. Other data centre topologies, such as VL2, incorporate centralised components which can provide similar information. The sender can thus choose an appropriate value for the *dupthresh* based on this information. For example, if a source sends its traffic via the core layer, then the *dupthresh* should be much higher, compared to when traffic crosses only a TOR switch.

In this paper we use the FatTree addressing scheme as the basis for setting *dupthresh*. Each source host infers the layer(s) of the network topology that its traffic would cross when transmitting data to a specific destination host, by examining the source and destination IP addresses. For example, when a connection needs to be established between nodes with IP addresses *10.0.1.1* and *10.0.1.2* then it can be inferred that both hosts are located within the same ToR switch; therefore the *dupthresh* value should not be changed from the default value of three. Traffic crossing the aggregate or core layers would require higher *dupthresh* values. In Section III-D, our evaluation confirms that our approach significantly decreases spurious retransmissions.

The knowledge of the end-host's location is essential but not sufficient to assign an appropriate value for the *dupthresh*; each end-host also needs to know the size of the network topology. For example, a network topology with 4 core switches requires a different value of *dupthresh* compared to a network topology with 8 core switches. Additionally, network switches may also support ECMP with a limited number of paths in each IP subnet (e.g. up to 16 equal-cost paths), therefore knowing these information is also important for deciding a precise value for duplicate ACK threshold.

III. EVALUATION

A. MMPTCP vs MPTCP

In Figure I we showed that although MPTCP performs well with long flows, it hurts completion time of short flows. The main reason for that is related to MPTCP's congestion control. In short, the congestion control algorithm does not completely remove traffic from the most congested paths. It removes traffic from congested paths exponentially and then places a small amount of new data on these paths until the network conditions are improved. If the amount of data on a subflow is very small then even experiencing a single packet drop may lead to the loss of the TCP ACK clock (i.e. no data packet can be sent since no ACK is received). In other words, a single packet drop from a subflow holds the entire MPTCP connection until that packet is recovered.

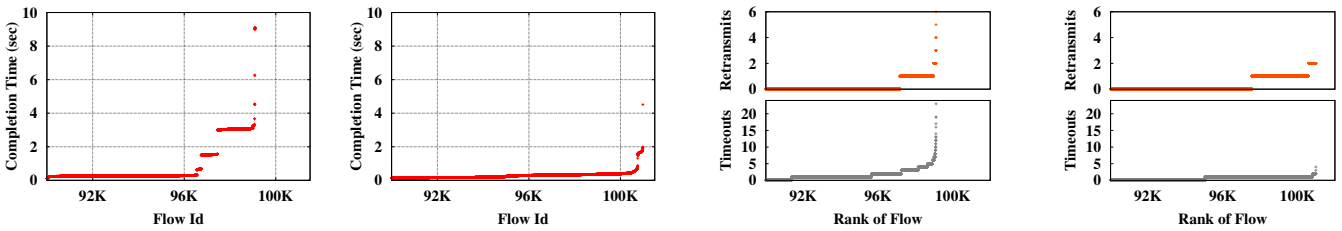
Keeping some traffic on subflows that experience congestion is a better approach than removing almost all of the traffic from those subflows when a flow is large [18]. For example, the Fully Coupled congestion control algorithm resets its congestion window to two segments in such a case [19], [20]. That is, MPTCP subflows can maintain their ACK clocks, and hence experience fewer timeouts, compared to Fully Coupled. However, neither approach is well-suited to short flows. To expand the above discussion, we have run a simulation in a 4:1 oversubscribed FatTree topology consisting of 512 servers for MPTCP with eight subflows and MMPTCP. One third of the servers run long (background) flows. The rest run short flows (70KBs each), which are scheduled by a central scheduler according to a Poisson arrival ($\lambda = 256$). All flows are scheduled based on a Permutation traffic matrix [5], [11]. The MMPTCP switching threshold is 100KBs. The results are depicted in Table I. The average short flow completion time for MPTCP

Transport Protocol	Short Flow Finish Time (mean/stdev)	Large Flow Goodput (mean/stdev)	Core Layer Utilisation (mean)	Core Layer Loss Rate (mean)
MPTCP	126/±425 ms	62.1/±19.7 Mbps	75.5 %	0.0077 %
MMPTCP	116/±101 ms	61.9/±20.0 Mbps	74.9 %	0.0076 %

TABLE I
MPTCP WITH EIGHT SUBFLOWS VS MMPTCP

is 126ms and the respective standard deviation is 425ms for 99103 completed short flows.[4] The high standard deviation indicates that there are some cases in which MPTCP performs far worse than the average. The average flow completion time for MMPTCP is 116ms and the respective standard deviation is 101ms for a total of 100980 completed short flows. This is a significant improvement (especially the standard deviation) which means that MMPTCP short flows maintain their ACK clock better than MPTCP with eight subflows when they

[4]We observed a high standard deviation in all runs of this simulation.



(a) MPTCP with eight subflows (b) MMPTCP (c) MPTCP with eight subflows (d) MMPTCP

Fig. 2. MPTCP with eight subflows vs MMPTCP. Short flow completion times (2(a) and 2(b)) and timeouts and fast retransmissions (2(c) and 2(d)).

experience loss events. The reason is that MMPTCP holds a single congestion window at initial phase of data delivery.

We also looked at the flow completion times, total fast retransmissions and timeouts of each individual short flow. Figures 2(c) and 2(d) illustrate the number of timeouts and fast retransmits for MPTCP and MMPTCP, respectively. It is clear that MPTCP suffers from excessive timeouts. Note that a few short flows experienced more than 20 timeouts and around ∼4K short flows experienced more than two timeouts during their lifetime. MMPTCP clearly outperforms MPTCP: it decreases the maximum timeouts and fast retransmissions from 25 to 4 and 6 to 2 respectively. The majority of short flows (more than 100K) experienced fewer than two timeouts (∼95K flows with no timeout). Figure 2(a) and 2(b) depict short flow completion times for MPTCP and MMPTCP, respectively. It is expected that with MPTCP a lot more short flows have very high completion times due to a larger number of timeouts compared to MMPTCP.

So far, we have shown that, unlike MPTCP, MMPTCP does not produce a heavy tail regarding short flow completion times, while it achieves high overall network utilisation and exceptional goodput for long flows. MMPTCP can therefore be deployed in existing data centres and used with all existing applications without relying on application information regarding flow sizes and potential deadlines. This is particularly important for data centre application designers who prefer not to consider underlying networking protocols when developing their applications.

B. MMPTCP vs TCP and PS

In this section we compare the performance of MMPTCP, TCP and PS using the same simulation setup as presented in subsection III-A. The results are depicted in Table II. TCP achieves the worst overall core utilisation and highest mean core loss rate. However, its mean flow completion time is lower than MMPTCP and MPTCP. PS achieves the lowest average short flow completion time and overall core loss rate with a high average goodput for large flows.

TCP performs badly with respect to network resources' utilisation because it transports data through a single path, therefore being unable to find and shift its traffic to the least congested paths. TCP gets trapped in a congested path and damages itself and other competing flows at bottleneck links along the path. This is the main reason that TCP achieves

Transport Protocol	Short Flow Finish Time (mean/stdev)	Large Flow Goodput (mean/stdev)	Core Layer Utilisation (mean)	Core Layer Loss Rate (mean)
PS	36.9/±38 ms	58.6/±18.2 Mbps	75.1 %	0.0001 %
TCP	64.3/±118 ms	38.5/±19.8 Mbps	44.7 %	0.0259 %
MMPTCP	116/±101 ms	61.9/±20.0 Mbps	74.9 %	0.0076 %
MPTCP	126/±425 ms	62.1/±19.7 Mbps	75.5 %	0.0077 %

TABLE II
SIMULATIONS WITH λ = 256

lower short flow completion times compared to MPTCP or MMPTCP, since a lot of unused capacity in the network is used by a majority of short TCP flows to complete their data deliveries in a short time frame. In other words, the inability of large TCP flows to utilise network resources provides headroom for short TCP flows to be completed faster. PS performs well in this experiment because it prevents the creation of any congestion in the core and aggregation layers by scattering packets of all flows in the network.

After this analysis, one might question the benefits of running MPTCP and/or TCP in today's data centres if PS can perform that well (as shown above). An important question here is why PS did not achieve the highest average goodput for large flows even though we observed very low loss rate in the network core. As it has been discussed in Section II-A, PS supports a single congestion window and, as a result, when a loss is detected, the rate of data transmission is halved. Unlike MPTCP, PS has no way to shift traffic to the least congested paths [5]. Therefore, it is expected that if PS coexists with other transport protocols, such as TCP and/or MPTCP, its performance will be significantly degraded. To examine this argument, we rerun the simulations above with λ = 2560 instead of 256. This simulation setup not only explains how the congestion control of each transport protocol behaves under highly dynamic traffic patterns but also explains how the congestion is actually produced by each transport protocol in the network. Furthermore, we designed a simulation, referred to as PS::TCP, which uses TCP for running short flows and PS for running long flows. PS::TCP helps to evaluate the performance of PS when it competes with non-PS flows such as TCP. The results are presented in Table III. PS::TCP achieves the lowest mean flow completion time, which entails the degradation of almost 10Mbps in the overall goodput and 15% less in the mean core utilisation. It also increases the mean core loss rate by 14 times compared to PS. TCP performs the worst in almost all comparisons except the

Transport Protocol	Short Flow Finish Time (mean/stdev)	Large Flow Goodput (mean/stdev)	Core Layer Utilisation (mean)	Core Layer Loss Rate (mean)
PS	40.5/±44.3 ms	52.9/±16.7 Mbps	76.8 %	0.0001 %
PS::TCP	29.7/±31.1 ms	42.5/±11.3 Mbps	61.9 %	0.0014 %
TCP	66.5/±150 ms	34.2/±18.1 Mbps	48.8 %	0.0576 %
MMPTCP	111/±127 ms	55.9/±18.7 Mbps	76.7 %	0.0105 %
MPTCP	148/±502 ms	55.0/±18.2 Mbps	75.9 %	0.0100 %

TABLE III
SIMULATIONS WITH λ = 2560

mean flow completion time. TCP also achieves the highest loss rates in the network core among all transport protocols. As it is expected, MMPTCP achieves a lower mean flow completion time and standard deviation compared to MPTCP with eight subflows. It also achieves the same overall network utilisation with MPTCP.

Analysis. The main reason that PS::TCP achieves a better overall flow completion time than PS is that large flows in PS::TCP are more susceptible to random packet drops, and hence they reduce their rates more frequently. When a buffer filled up their packets most likely are in the tail of the queue since large flows randomly spread their packets via all possible paths. The consequence of such rate reductions is to decrease some traffic throughout the network (from all queues). This helps many short flows to complete their data delivery without experiencing any collision and with less queuing delay.

The above experiment justifies that PS is very sensitive to network congestion and when it is used for handling large flows it hurts their connection throughputs, and consequently the overall network utilisation.

C. Effects of Hotspot

The main goal of this study is to understand how each transport protocol reacts when hotspots exist in the network. These hotspots may occur for several reasons in modern data centres, including: (1) contention between traffic flowing from the Internet to data centres, (2) hardware failures or cable faults, (3) uneven load in some servers. In order to model hotspots in the core layer, we modified the drop tail queue size of hotspot links from 100 to 10 packets.[5] To select links under the hotspot, we select all the links of some randomly selected core switches. In this way, we can monitor the hotspot areas by simply monitoring each core switch under the hotspot.

Simulations in this section were conducted in various number of hotspot core switches, ranging from 20% to 60% of total core switches (we refer to the percentage of cores under hotspot as 'hotspot degree'). The network topology used is a 4:1 oversubscribed FatTree topology. The traffic matrix used is Permutation and the value of λ is 2560.

It is expected that by increasing the hotspot degree, the overall network utilisation will decrease, the overall short flow completion time and mean loss rate will increase with all transport protocols. We aim to see how each transport protocol follows this trend.

[5]To select a size for the drop tail queue, we examined various queue sizes, ranging from 10 to 50 packets, and it turned out that 10 packets can best show the distinctions between the behaviour of the different transport protocols.

Figure 3(c) shows the mean goodput achieved by large flows of each transport protocol under various hotspot degree. It is noticeable that TCP and PS::TCP almost achieve the worst and MMPTCP achieves the best overall goodput for large flows. This is another highlighted experiment to show the weakness of PS and the strength of MMPTCP in handling congestion.

Figure 3(b) shows that as the hotspot core switches increases, MMPTCP behaves consistently and achieves the highest mean core utilisation at all hotspot degrees.

At the hotspot degree of 60, the TCP short flows achieves the mean flow completion time of 80.5ms with a standard deviation of 214ms.[6] This implies that hotspots in TCP mostly exert influence on short TCP flows because large TCP flows have already shown their inability to use network resources efficiently, even without any hotspot link, due to ECMP hash collisions. Figure 3(a) illustrates the mean core loss rate achieved by each transport protocol. MMPTCP achieves the lowest mean loss rate at all hotspot degrees. By increasing the percentage of hotspot cores, the mean loss rate of PS::TCP and TCP increases significantly as both simulation setups use TCP for running short flows. The completely opposite result is achieved with MMPTCP and PS because both simulations use the PS protocol for handling short flows.

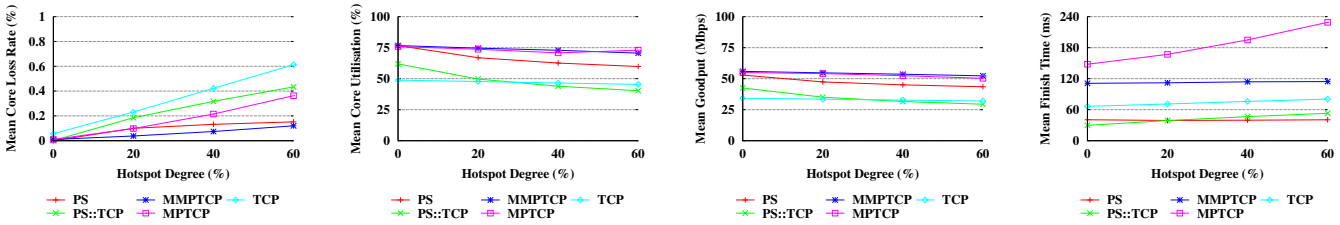
The intuition following this experiment is that the burstiness of data centre traffic, which arises from short TCP flows, is smoothed by using MMPTCP. In other words, the TCP protocol for handling short flows not only increases congestion but also fails to handle it gracefully. However, MMPTCP not only prevents possible congestion by scattering packets in the network, but also handles it effectively by shifting traffic away from congested areas, after switching to MPTCP. This is the main reason that MMPTCP achieves the lowest loss rate at various hotspot levels compared to other simulation setups. Even PS is not capable of dealing with hotspots effectively since it cannot detect them. These observations are highlighted in Figure 3(a).

D. MMPTCP and Duplicate ACK Threshold

In this section, we review the adjustment of the *dupthresh* value during the initial phase of MMPTCP. We then examine our novel solution for *preventing* spurious retransmissions due to packet reordering, as described in subsection II-D.

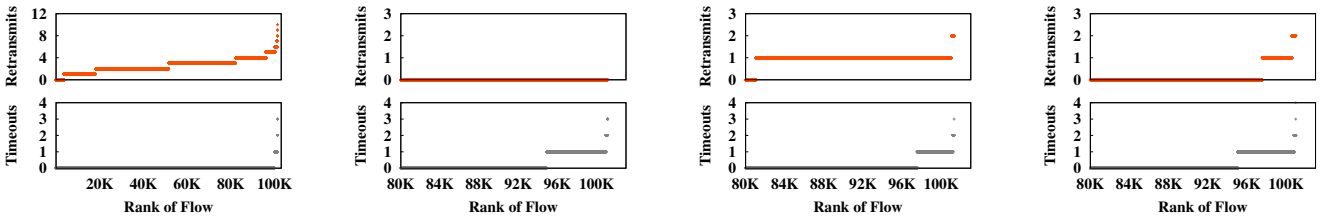
To explore the effect of packet reordering in our model of data centre, we conducted a series of simulations with a varying *dupthresh* value ranging from 3 to 23. Simulations were conducted in a FatTree topology with 128 servers running short and large MMPTCP flows. 33% (42) of servers send background flows and the remaining 67% (86) of servers send short flows (λ = 256). The result is shown in Figure 4. It is clear that the default duplicate ACK threshold of three achieves the worst average flow completion time (158ms). However, by increasing the value of that, the average flow completion time decreases significantly to a *dupthresh* of eight. Thereafter

[6]TCP achieves the highest standard deviation compared to other transport protocols. Due to a lack of space, we removed these information, but we confirmed them.



(a) (b) (c) (d)

Fig. 3. All simulation setups under varied hotspot core switches. MMPTCP achieves the lowest mean core loss rate, the highest mean large flow goodput and the highest mean core utilisation at all hotspot degrees.



(a) *dupthresh* of 3 (b) *dupthresh* of 23 (c) *dupthresh* of 9 (d) Auto *dupthresh*

Fig. 5. Timeouts and fast retransmissions for each individual short flow in various *dupthresh* values and our solution (auto *dupthresh*).

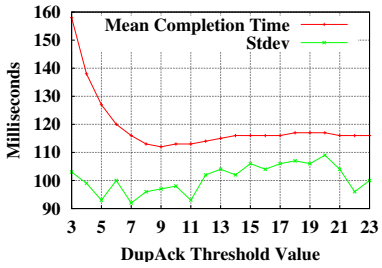


Fig. 4. Duplicate ACK threshold value effect on short flow completion time

the result remains unchanged with a little fluctuation in the standard deviation. To get a better grasp of the problems, we look at the number of fast retransmissions and timeouts experienced by each short flow. Figures 5 shows the result for a *dupthresh* of 3, 23, 9 and auto (our solution). At one extreme, which is related to a *dupthresh* of three, we observed the highest fast retransmission hits and lowest timeout hits (5(a)). At another extreme, which is related to a *dupthresh* of 23, we observed no fast retransmission hits and high timeout hits (5(b)). The best performance is observed when the threshold is set to nine (5(c)). The majority of flows were completed with no fast retransmission (∼81K) or with only one (∼19K); a few flows experienced two fast retransmissions. The results of this experiment do not lead to any concrete value for the *dupthresh* since they are only valid for this particular network setup. By altering the network topology, e.g. its size or traffic matrices, the performance of selected *dupthresh* value might become unsatisfactory. Another issue with selecting a single value for the *dupthresh* is that most of the traffic may be localised in ToR switches in which they do need to have a *dupthresh* larger than the default value of three.

Figure 5(d) shows the result of simulating an auto *dupthresh* with the same simulation setup as Figure 4. Auto *dupthresh* significantly decreases the number of spurious retransmission due to packet reordering by adjusting the value of *dupthresh* based on topology-specific information. This can be observed by comparing the line one at retransmit plots in Figure 5(c) and 5(d). However, by comparing the line one at timeout plots one can observe that the auto *dupthresh* slightly increases the number of timeouts compared to *dupthresh* of 9.[7]

Increasing or adjusting the *dupthresh* value is a tricky task as the TCP New Reno sender could lose its ACK clock, especially when the value of the *dupthresh* is larger than the congestion window. If any packet gets dropped in such scenarios, TCP needs to wait for a retransmission timer to be triggered. For example, in the above simulations 85% of network flows traverse the network core due to the Permutation traffic matrix. This implies that with auto *dupthresh* solution a majority of short flows set their *dupthresh* value to 19. If any segment gets dropped at the first five RTTs, either at the beginning of data transmission or after any timeout event, the corresponding subflow should wait until its retransmit timer is triggered. Therefore, the large *dupthresh* value is the main reason that auto *dupthresh* achieves a slightly higher timeout hits compared to *dupthresh* of 9. In order to improve the performance of auto *dupthresh* and hence MMPTCP in such scenarios, we propose to use the TCP Limited Transmit mechanism in the initial phase of MMPTCP.

E. MMPTCP and Limited Transmit

Limited Transmit (LT) is an enhancement to TCP loss recovery and attempts to prevent RTOs, especially when the

[7]We used this solution for all simulations conducted with MMPTCP and PS in this paper.

congestion window size is very small [17], [21]. LT allows a TCP sender to transmit new segments only upon arrival of the first two duplicate ACKs on a segment, i.e. before the fast retransmission is triggered.

We modified this algorithm so that a TCP sender allows new segments to be sent before fast retransmission is triggered regardless of the *dupthresh* value. For example, if *dupthresh* is 19 then a TCP sender allows to send 18 new segments before triggering the fast retransmission. In this way, a sender can prevent timeouts when a packet gets dropped and congestion window is smaller than *dupthresh*. We have integrated this new algorithm into the initial phase of MMPTCP.

To evaluate the performance of MMPTCP with LT, we designed a new version of MMPTCP with activated LT, referred as MMPTCP$_{\text{LT}}$. We then compared it with MMPTCP in a 2:1 oversubscribed FatTree topology with 256 servers running the Permutation traffic matrix (λ = 256). 53% (135) of servers send background flows and the remaining 47% (121) of servers send short flows. The first short flow schedules 500ms after simulation starts in order to let large flows become stable. Table IV depicts the results. MMPTCP$_{\text{LT}}$ significantly improves mean flow completion time and standard deviation of short flows without damaging overall network utilisation. We

Transport Protocol	Short Flow Finish Time (mean/stdev)	Large Flow Goodput (mean/stdev)	Core Layer Utilisation (mean)	Core Layer Loss Rate (mean)
MMPTCP	98.9/±74.8 ms	72.9/±17.3 Mbps	72 %	0.0053 %
MMPTCP$_{\text{LT}}$	89.1/±67.2 ms	72.9/±18.0 Mbps	72 %	0.0051 %

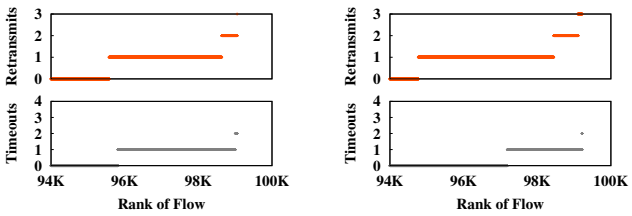
TABLE IV
MMPTCP COMPARED TO MMPTCP$_{\text{LT}}$

also look at the total fast retransmissions and timeouts in each individual short flow in Figure 6. As expected, MMPTCP$_{\text{LT}}$ slightly increases the number of fast retransmission in favour of decreasing the number of timeouts compared to MMPTCP. In fact, MMPTCP$_{\text{LT}}$ protects short flows from losing their ACK clocks when a high *dupthresh* value is used (e.g. 19). Therefore, the completion times of a majority of short flows is decreased with MMPTCP$_{\text{LT}}$.

It is argued that LT is an essential mechanism for *preventing* TCP from losing its ACK clock, especially when *dupthresh* is adjusted automatically and is large [17]. However, LT becomes more aggressive as the *dupthresh* value increases. This may be less critical for MMPTCP because its short flows use all possible paths in data delivery. However, even with MMPTCP, if there is a hotspot in the access layer then this aggressiveness becomes important and may hurt other competing flows in such a case. Further research is required in order to understand how LT should be used when *dupthresh* is very large.

F. MMPTCP and Switching Threshold

In this section, we investigate the effects of the MMPTCP switching point on the completion time of short flows when the size of short flows is lower or higher than a switching point. In doing so we conducted a range of simulations with varying short flow sizes over various switching points. Table V shows the results. It is clear that changing the switching threshold



(a) MMPTCP (b) MMPTCP$_{\text{LT}}$

Fig. 6. Timeouts and fast retransmissions (MMPTCP against MMPTCP$_{\text{LT}}$)

Short Flow Size (KB)	Switching Threshold (KB)	Short Flow Finish Time (mean/stdev)	Large Flow Goodput (mean/stdev)	Core Layer Utilization (mean)	Core Layer Loss Rate (mean)
70	100	98.9/±74.8 ms	72.9/±17.3 Mbps	72.9 %	0.0053 %
70	300	98.4/±79.0 ms	72.7/±18.5 Mbps	71.7 %	0.0043 %
70	500	97.7/±74.8 ms	72.8/±18.2 Mbps	71.9 %	0.0041 %
70	1000	98.5/±75.1 ms	72.6/±18.1 Mbps	71.7 %	0.0037 %
600	100	324.8/±198.0 ms	67.9/±16.2 Mbps	74.4 %	0.0080 %
600	300	321.8/±194.7 ms	67.6/±17.4 Mbps	74.2 %	0.0068 %
600	500	312.5/±196.7 ms	67.8/±17.0 Mbps	74.4 %	0.0064 %
600	1000	325.3/±195.8 ms	67.7/±17.0 Mbps	74.3 %	0.0062 %

TABLE V
MMPTCP SWITCHING THRESHOLD SENSITIVITY

does not exert any negative effect on the completion time of short flows since the results for a flow size (e.g. 70KBs) with different switching thresholds are extremely consistent. Due to lack of space, we have only shown two different flow sizes in Table V. But we have experimented with flow sizes ranging from 50KBs to 1MB over varying switching thresholds ranging from 100KBs to 10MBs. We observed very consistent results in all of our simulations. This is a very important outcome because it is very likely that some short flows in a data center have larger sizes than a switching threshold.

IV. DISCUSSION

During our evaluation, we realised that employing TCP congestion control during the initial phase of MMPTCP is an overkill approach. Because the congestion window operates over multiple paths, when a congestion signal originated from a random link at the network core, it is overkill to react to that congestion signal by halving the sending rate, since there is no congestion on any of the other paths. However, if a congestion signal comes from a bottleneck link at the access layer, then the reaction of TCP congestion control is correct. The research question here is how can we distinguish these two signals and react appropriately. Our hypothesis is that reacting to congestion proportionally to the extent of congestion will allow detection of these two signals. We thus believe that employing the DCTCP-link congestion control could be a viable solution for distinguishing these two signals. If a congestion signal comes from random links at the network core then the proportion of congestion signals, during one RTT, is very low so DCTCP does not reduce its sending rate. However, if it is from a bottleneck link at the access layer, DCTCP reacts similarly to TCP. Further investigation is required to determine best practices, parameter adjustments, and so on.

MMPTCP is capable of utilising multi-homed network topologies. Unlike MPTCP, MMPTCP is capable of delivering all network flows via all available network interface devices. This feature potentially allows the TCP Incast problem to be addressed by adding more interface devices to end-hosts. We plan to conduct further research on the performance of MMPTCP over multi-homed topologies, such as Dual-Homed FatTree (DHFT) [5].

In this paper, we evaluated MMPTCP with TCP NewReno, which is a widely deployed TCP version. However, TCP NewReno is not an ideal solution when packet reordering is the norm. We explored a solution for preventing packet reordering by increasing *dupthresh* in order to postpone the triggering of the fast retransmission mechanism. However, any solutions that attempt to increase the value of *dupthresh* may increase timeouts when a packet gets dropped while the congestion window is smaller than *dupthresh*. To address this, we activated TCP limited transmit during the initial phase of MMPTCP. We believe increasing *dupthresh* and coupling it with limited transmit is the right approach for preventing spurious retransmissions in modern data centres, but further research is required into the degree to which limited transmit should react to duplicate ACKs. We also plan to investigate how DSACK will improve the performance of MMPTCP, as it can help to detect and mitigate spurious retransmissions.

Advanced QoS features have become increasingly available in data centre switches [22]. Our hypothesis is that if packets of the initial phase of MMPTCP are marked high priority and routed through different queues, then MMPTCP effectively helps latency-sensitive short flows to meet their deadline. The packets of short flows are thereby routed from a different queue to the large flows so that the chance of random packet drop significantly decreases, especially at the network core.

V. CONCLUSION

In this paper, we first conducted an in-depth study of MPTCP for short flows. We observed that MPTCP is ill-suited to handle them. A fraction of short flows complete their flows with a long delay because they incur excessive timeouts. We then proposed MMPTCP as a means to address this problem. Our extensive experimental evaluation in simulated FatTree topologies showed that MMPTCP is practical and decreases flow completion time for short flows while retaining high goodput for large flows over MPTCP with a fixed number of subflows. We also observed that MMPTCP not only reacted to congestion gracefully but also prevented it to a great extent, thereby significantly decreasing the overall loss rate of all links in the network.

One of MMPTCP's challenges is to prevent, detect and react to spurious retransmission due to packet ordering during its initial phase of delivery. In this paper, we proposed a novel approach for preventing out-of-order packets which is to set the value of *dupthresh* based on topology-specific information. Our solution is based on the FatTree IP addressing scheme as it allows us to locate end-hosts according to their IP address. That is, the *dupthresh* is adjusted according to the destination IP address of a flow at connection establishment. Our investigation showed that adjusting *dupthresh* in this way significantly prevents spurious retransmission. Our approach is also practical in other data center topologies such as VL2.

We conclude that MMPTCP is rapidly deployable in existing data centres as it coexists with other transport protocols and only requires existing data centre technologies such as ECMP. It can handle all network flows without high-level information from application layers (e.g. flow sizes and deadlines). It decreases the bursty nature of data centres by leveraging parallel paths for delivering short flows.

REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proc. of SIGCOMM 2008*.

[2] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proc. of SIGCOMM 2011*.

[3] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: a high performance, server-centric network architecture for modular data centers," in *Proc. of SIGCOMM 2009*.

[4] C. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm," RFC 3782, 2004.

[5] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, and M. Wischik, D.and Handley, "Improving Datacenter Performance and Robustness with Multipath TCP," in *Proc. of SIGCOMM 2011*.

[6] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proc. of SIGCOMM 2010*.

[7] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware Datacenter TCP (D2TCP)," in *Proc. of SIGCOMM 2010*.

[8] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better Never than Late: Meeting Deadlines in Datacenter Networks," in *Proc. of SIGCOMM 2011*.

[9] A. Ford, C. Raiciu, M. Handley, S. Barré, and J. Iyengar, "TCP Extension for Multipath Operation with Multiple Addresses," RFC 6824, 2013.

[10] M. Kheirkhah, I. Wakeman, and G. Parisis, "Multipath-TCP in ns-3," in *Proc. of WNS3 2014*.

[11] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proc. of NSDI'10*.

[12] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the Impact of Packet Spraying in Data Center Networks," in *Proc. of IEEE INFOCOM 2013*.

[13] L. Valiant, "A Scheme for Fast Parallel Communication," *SIAM journal on computing*, vol. 11, no. 2, pp. 350–361, 1982.

[14] S. Floyd, J. Mahdavi, M. Mathis, and M. Podolsky, "An Extension to the Selective Acknowledgement (SACK) Option for TCP," RFC 2883, 2000.

[15] J. Mahdavi, M. Mathis, S. Floyd, and A. Romanow, "TCP selective acknowledgment options," no. 2018, 1996.

[16] S. Floyd and T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 6582, 2002.

[17] M. Zhang, B. Karp, S. Floyd, and L. Peterson, "RR-TCP: A Reordering-Robust TCP with DSACK," in *Proc. of ICNP 2003*.

[18] D. Wischik, C. Raiciu, and M. Handley, "Balancing Resource Pooling and Equipoise in Multipath Transport," in *Proc. of SIGCOMM 2010*.

[19] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley, "Multi-path TCP: a joint congestion control and routing scheme to exploit path diversity in theInternet," *IEEE/ACM Trans*, vol. 14, no. 6, 2006.

[20] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, Implementation and Evaluation of Congestion Control for Multipath TCP," in *Proc. of USENIX NSDI 2010*.

[21] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's Loss Recovery Using Limited Transmit," RFC 3042, 2001.

[22] D. Zats, T. Das, P. Mohan, and R. Katz, "DeTail: Reducing the Flow Completion Time Tail in Datacenter Networks," in *Proc. ACM SIGCOMM 2012*.