Reducing transport latency for short flows with multipath TCP

Pingping Dong , Wenjun Yang , Wensheng Tang , Jiawei Huang , Haodong Wang , Yi Pan , Jianxin Wang

1. Introduction

Live and interactive applications are often sensitive to latency, such as web transfers, video streaming and online gaming (Yedugundla et al., 2016). In such applications, the user's experience is affected significantly when data delivery is delayed. For instance, Google measured a 25% reduction in the number of searches done by users, as a result of adding 500 ms to web search time (Hwang et al., 2015).

Multipath TCP (MPTCP) is an ongoing effort by the Internet Engineering Task Force (IETF) (Ford et al., 2013). By allowing the use of multiple network paths for a single data stream, MPTCP increases robustness upon path failure, and potentially achieves higher end-to-end throughput (Habib et al., 2016). It is a backward-compatible TCP extension and has been widely applied in SiRi of iOS7, Samsung Galaxy S6, the Netscaler load balancing schemes of Citrix, and so on.

The effectiveness of MPTCP for long flows have been proved by prior work (Dong et al., 2016; Kheirkhah et al., 2016; Peng et al., 2016), benefit from simultaneous transmission of data over multiple interfaces. It can efficiently pool networks resources for long TCP flows. However, MPTCP occasionally becomes inefficient while handling short flows (Kheirkhah et al., 2016). The reasons are twofold.

First, MPTCP leads to increased delays when the subflows have heterogeneous path characteristics (Yedugundla et al., 2016; Cordero, 2016). On one hand, the path heterogeneity causes packet reordering or even head-of-line blocking. For example, the packets sent over the fast subflow can fill up the receiver buffer while waiting for the data

transferred in the slow subflow. On the other hand, MPTCP may choose a slow path if the congestion window of the fast path is not available, resulting in a longer flow completion time.

Second, the majority of the short flows are often quite small and more than 40% of the web traffic are short flows with the size smaller than 1 MB (Barik et al., 2016). For these short flow transmissions, TCP will likely never leave the slowstart (SS) phase and the congestion window of an individual subflow may be very small over its lifetime as packets are spread across all available subflows in MPTCP. In this case, 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 (Kheirkhah et al., 2016), resulting in excessive latency for the delay-sensitive applications.

Based on the above observation, we conclude that the number of the subflows utilized by a flow has a significant influence on MPTCP performance. Increasing the number of subflows is beneficial to the Goodput of long flows, but it is harmful to flow completion time of short flows. If a predefined number of subflows is used for all types of flows, then the performance of MPTCP can be significantly degraded.

To mitigate this problem and improve the performance of both short flows and long flows with MPTCP, we propose DMPTCP in this paper. DMPTCP aims to find the set of the subflows automatically suitable for a certain application based on the MPTCP analytical model to achieve both performance improvement and latency reduction. The main contributions of this paper are as follows

- Based on the in-depth study of MPTCP for short flows, we propose a path selection algorithm, namely DMPTCP for concurrent multipath transfer. To effectively use all available paths with the guarantee to meet the completion time of short flows, DMPTCP finds the set of subflows for a certain application by estimating the total data amount (i.e., N) that can be sent on the faster subflows simultaneously before the packet arrival time over the slower subflow.
- A key challenge in the proposed DMPTCP algorithm is the accuracy of the transferred data amount estimation over fast paths. To this end, this paper utilizes the idea of TCP modeling that takes into account path characteristics, namely, the congestion window, the round trip time, the packet loss rate, and the MPTCP's four congestion control algorithms over each subflow, e.g. slow start, the coupled congestion avoidance, fast retransmit, as well as fast recovery.
- We validate the effectiveness of DMPTCP by conduct extensive experiments. The simulations are based on our DMPTCP implementation in Network Simulator-3 (ns3). The results demonstrate that DMPTCP is practical and reduces flow completion time for short flows while retaining high Goodput for large flows as MPTCP. We also implement DMPTCP on Linux kernel by setting up a testbed consisting two subflows. Compared with the MPTCP default scheduler minRTT, the decreased completion time is up to 35.73% for the short flows while for long flows, this revenue is up to 21.3%.

The rest of the paper is organized as follows. The design motivation of DMPTCP is presented in Section 2. Section 3 discusses the related works. Section 4 describes the details of DMPTCP and presents the model analysis. We respectively evaluate DMPTCP with both NS3 and the Linux testbed in Section 5 and 6. Finally, Section 7 concludes the paper.

2. Motivation

In this section, we conduct empirical studies to analyze the root reason why current MPTCP protocol fails to provide satisfactory performance and present the design objectives.

MPTCP is a set of extensions to TCP developed by the IETF MPTCP working group to achieve efficient resource usage and improve user experience by utilizing multiple paths simultaneously. However, this

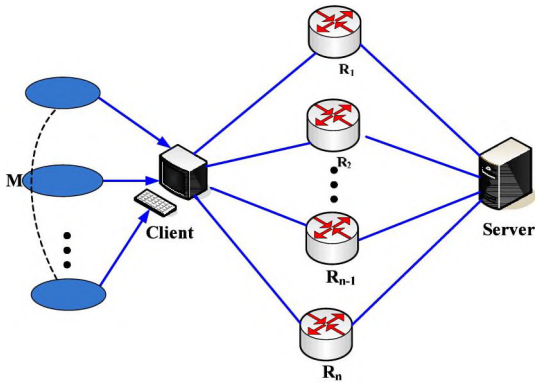


Fig. 1. The topology of the NSL (Non-shared-links) scenario where all subflows have disjoint paths.

potentially impairs the transmission performance for short flows especially in heterogeneous scenarios. Firstly, a slow path can be taken into account in the utilization. Further, path heterogeneity leads to packet reordering. Besides, the congestion window over each subflow can be very small as the data is spread over all subflows in MPTCP, which may lead to RTOs when packet loss occurs. All of these issues together cause the increased end-to-end delay and path bandwidth underutilisation.

In order to illustrate the issues, we conduct our experiments with the topology shown in Fig. 1 where M applications run between the client and server which are connected through N disjoint routers. Fig. 2 shows the average completion time of 30 concurrent short flows transfer across two subflows. The sizes of these short flows obey the Pareto distribution with an average value of 57 KB as described in Section 5. One of the subflows has a fixed RTT of 10 ms while the other subflows have their RTTs from 10 ms to 200 ms as depicted in the figure.

According to Fig. 2, when the two path are homogeneous or have little difference (i.e., 10 ms, 20 ms), MPTCP outperforms TCP as two paths are used compared to the single-path TCP. However, as the RTT of the second subflow increases, the performance of MPTCP is getting worse. This phenomenon is also reported by the prior research (Ferlin et al., 2016b; Oh and Lee, 2015; Yedugundla et al., 2016). Fig. 3(a) illustrates a more severe scenario as MPTCP has even worse performance than TCP, which is caused by RTO as further revealed in Fig. 4 and Fig. 5. When the random packet loss is present, as depicted in Fig. 3(b)), the MPTCP performance degradation is more significant.

As follows from Fig. 4 that when the flow size is small, the data amount spread over each subflow is quite small. Then, the lost packets

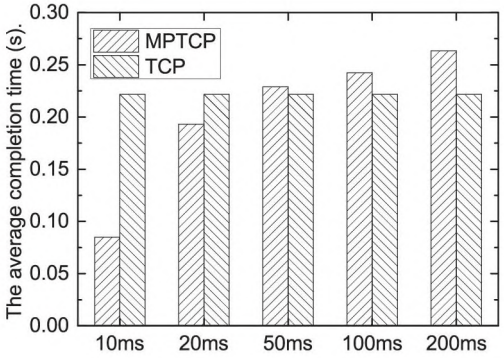
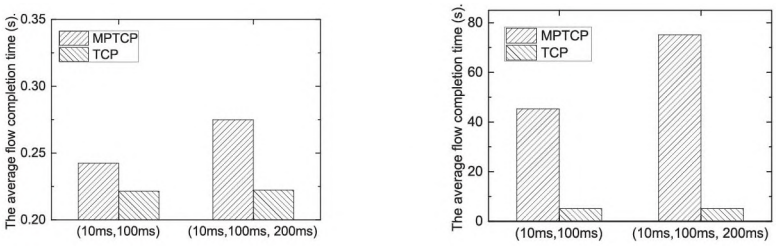


Fig. 2. The average completion time of MPTCP when the RTT of one subflow is fixed at 10 ms and the other varies from 10 ms to 200 ms.



(a) No random packet loss. (b) 0.05% random packet loss rate.

Fig. 3. The average completion time of MPTCP with varying number of subflows whose RTT is illustrated in the horizontal x-axis.

cannot recover from fast retransmission and RTO occurs frequently as shown in Fig. 5, resulting longer flow completion time for short flows, which is consistent with the results shown in the literature (Kheirkhah et al., 2016).

Based on the above analysis, we conclude that the amount of data and the path heterogeneity are the main factors determining the performance of MPTCP. These observation motivates us to design a novel approach to select appropriate subflows for each application to improve the performance of MPTCP in heterogeneous network scenarios. In the rest of this paper, we present our DMPTCP as well as its performance validation with extensive experiments.

3. Related work

Multipath TCP (MPTCP) increases bandwidth and robustness by using multiple network interfaces and multiple paths in parallel (Peng et al., 2013). Its performance is certain to be influenced by many factors (Paasch et al., 2014). Congestion control and path scheduling are two important aspects and have drawn considerable research attention.

The congestion control algorithm of MPTCP is responsible for adaptively adjusting the transmission rate of each subflow in an attempt to shift traffic from more congested paths to less congested ones, thus improving the throughput and link utilization. So far, several MPTCP congestion control algorithms have been proposed: LIA (Linked Increases Algorithm (Wischik et al., 2011)), OLIA (Opportunistic Linked-Increases Algorithm (Khalili et al., 2013)), Balia (Balanced Linked Adaptation (Peng et al., 2016)), wVegas (Weighted Vegas (Cao et al., 2012)) and mVeno (Dong et al., 2016).

Next, the scheduling policy is designed for distributing data packets over multiple paths, given the congestion window size of each path by the congestion control algorithm. The current default scheduler, Lowest-RTT-First (minRTT), first assigns packets to the fastest subflow until its congestion window is filled with data. Then packets are allocated to the other subflows. Although this policy works better than the Round-Robin policy in many cases, recent research suggested that RTT can result in suboptimal resource utilization, particularly when the paths are dissimilar (Raiciu et al., 2012; Chen et al., 2013; Oh and Lee, 2015). This heterogeneity results in packet reordering, which leads to Head-of-Line (HoL) blocking, and subsequently increases out-of-order (OFO) buffer at the receiver side and ultimately reduces the Goodput. Various enhanced scheduling schemes, like BLEST (Blocking Estimation-based MPTCP Scheduler) (Ferlin et al., 2016b), CP scheduling (Constraint-based proactive scheduling) (Oh and Lee, 2015), DAPS (Delay-Aware Packet Scheduler) (Kuhn et al., 2014; Sarwar et al., 2013), OTIAS (Out-of-order Transmission for In-order Arrival Scheduler) (Yang et al., 2014), HSR (Highest Sending Rate), LWS (Largest Window Space) and LTS (Lowest Time/Space) (Kimura et al., 2017) have been proposed to fully utilize all paths.

Although the MPTCP performance can be enhanced with these proposals, it still suffers from high interactive delay for short flows compared with regular TCP when the flow size is small, e.g., only hundreds of KB (Kheirkhah et al., 2016; Yedugundla et al., 2016). The reason lies in that MPTCP may still utilize the slow path when there is no available congestion window in the fast path. Besides, for these short flows, the amount of data is so small that it may be transmitted within TCP's initial window, given multiple concurrent connections (Hwang et al., 2015). In this case, the congestion window is so small that even a single packet loss may force an entire connection to wait for an RTO to be triggered because this lost packet cannot be recovered through fast retransmission (Kheirkhah et al., 2016). It has been proved that, due to the increased retransmissions, TCP completes the transmission approximately 50 ms faster than MPTCP of a 300 KB file transfer across a 2.5 Mbps shared bottleneck (Barik et al., 2016).

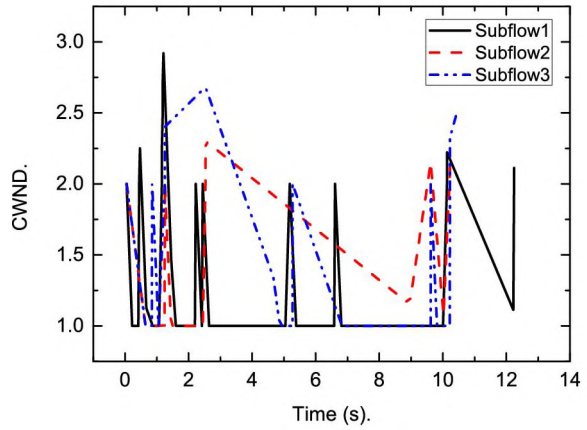


Fig. 4. The congestion window over each subflow of MPTCP when the application size is 36927 Bytes.

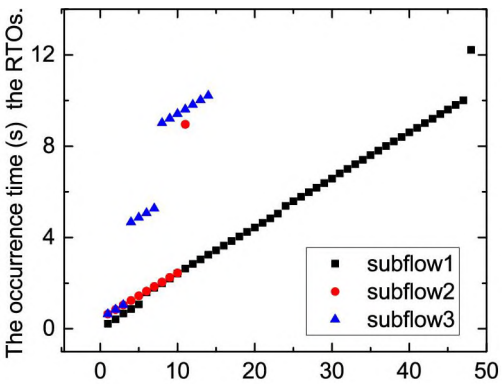


Fig. 5. The occurrence time of retransmission timeout over each subflow with MPTCP when 0.05% random packet loss exists.

To solve these problems, J. Hwang and J. Yoo (Hwang and Yoo, 2015) propose to freeze the slow path when the delay difference between the slow and fast paths is significant to guarantee that the short flow can be transmitted quickly via the fast path. S. Ferlin et al. (2016b) proposed a proactive scheduler which decides at packet scheduling time whether to send packets over the slow subflow or not. However, both these two mechanisms only consider the situation where the number of the subflows is two. M. Kheirkhah et al. propose AMPTCP (Kheirkhah et al., 2016) to achieve the high throughput for large flows and keep the low latency for short flows. This algorithm 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. After a specific amount of data is sent, AMPTCP switches to a regular MultiPath TCP mode, efficiently handling long flows by a separate congestion window for each subflow. Previous research has argued that packet scatter (PS) can achieve perfect performance if traffic load is equal among the servers (Raiciu et al., 2011). This situation can be satisfied in data center networks with a uniform network topology, such as FatTree or VL2. However, in real wide area networks, the network conditions can be quite different. 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. The arrangement may drastically reduce the throughput. Thus, how to improve the performance of short flows while maintaining the throughput of long flows with MPTCP in real WAN is a challenge.

4. The proposed algorithm

The goal of the DMPTCP algorithm is to evaluate all the available paths and identify the subflows that should be used to improve the MPTCP performance for both short flows and long flows. In this section, we first describe the MPTCP model that estimates the amount of data transmitted in each subflow for a given RTT. Then, we present DMPTCP and analyze the effectiveness of the analytical model.

4.1. Analytical model for data transmission

In this subsection, we present the analytical model for the MPTCP data transmission to study its throughput performance in order to predict the amount of data that can be transmitted. This proposed MPTCP model utilizes MPTCP characteristics of each TCP subflow including the congestion window, the round trip time and the packet loss rate. Meanwhile, MPTCP's four intertwined congestion control algorithms, i.e. slow start, the coupled congestion avoidance, fast retransmit, and fast recovery, are also taken into consideration.

According to the MPTCP's protocol stack as shown in Fig. 6, each TCP-based subflow has its own congestion window and the congestion control in MPTCP is performed at the subflow level. However, to keep the bottleneck fairness while competing with single path TCP flows, MPTCP uses the coupled congestion control algorithm by modifying the additive increase during the congestion avoidance phase. The congestion window increasement is calculated according to the network condition of all the subflows belonging to an MPTCP connection rather than one in the regular TCP. Several approaches to handle this issue are available, such as, LIA (Linked Increases Algorithm (Wischik et al., 2011)), OLIA (Opportunistic LIA (Khalili et al., 2013)), Balia (Balanced LIA (Peng et al., 2016)), wVegas (Weighted Vegas (Cao et al., 2012)) and mVeno (Dong et al., 2016). Coupled congestion control only applies to the increase phase of the congestion avoidance state, specifying how the congestion window inflates upon receiving an acknowledgement. Other phases are the same as in standard TCP.

In the current implementation in ns3, the default MPTCP congestion control algorithm is LIA, a.k.a. the RTT Compensator. In RTT Compensator, the congestion windows are adapted as follows.

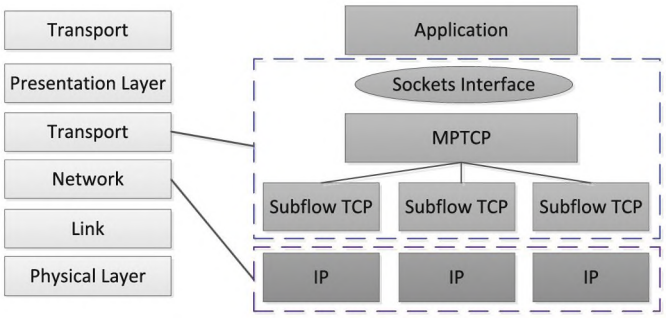


Fig. 6. The protocol stack of MPTCP.

- Each ACK on subflow r, increase the window w_r by $\min(\alpha/w_{total}, 1/w_r)$.
- Each loss on subflow r, decrease the window w_r by $w_r/2$.

Here

$$\alpha = \hat{w}_{total} \frac{\max_r \hat{w}_r/RTT_r^2}{\left(\sum_r \hat{w}_r/RTT_r\right)^2}. \quad (1)$$

RTT_r is the round trip time as measured by subflow r. w_r is the current window size on path r and $\hat{w}_r$ is the equilibrium window size on path r, and similarly for w_{total} and $\hat{w}_{total}$.

Based on the above analysis, we can obtain the congestion control process of MPTCP on each subflow. Like regular TCP, it goes from the slow-start phase to the congestion avoid phase when a loss event occurs or the congestion window reaches the slow start threshold. The congestion window size is doubled in the slow start phase and increased linearly according to Eq. (1) in the congestion avoidance phase during each RTT if no packet loss occurs. TCP both considers retransmission timeout (RTO) and duplicate ACKs as packet loss events. If duplicate ACKs are received, the sender reduces its congestion window to one-half of the current window size and SSThresh is set to be either *cw* or two MTUs (whichever is greater). If an ACK times out (RTO timeout), slow start is used where TCP reduces the congestion window to 1 MSS.

According to the behavior of MPTCP described above, we propose the analytical model based on prior work (Ma et al., 2007) to predict the data amount that can be transferred for a certain RTT round. In this multipath transfer model, each MPTCP connection consists of a set of subflows R, each of which may take a different route. Every subflow $r \in R$ maintains its own congestion window, slow start threshold, packet loss probability and round trip time. An MPTCP sender stripes packets across these subflows as the space in the subflow windows become available.

We model the MPTCP transmission over each subflow in terms of *round* in the paper as that in (Ma et al., 2007), where a round starts when the sender transmits the packets in its congestion window and ends when the last ACK is received in the current window. For simplicity, we assume that the duration of a round is equal to a round trip time (RTT) and independent of the congestion window. We also assume that the packet transmission time is much smaller than the round trip time.

The variables used in this model are listed in Table 1 where *cw* and *sst* represent the congestion window and SSThresh, respectively.

We use the variables $C_r(i)$ and $T_r(i)$ to denote the accumulated number of successful data transmissions and the accumulated transmission time until the i^{th} round, respectively. Then, the throughput $TH_r(i)$ obtained until the ith round of subflow r can be expressed as:

$$\mathrm{TH}_r(i) = \frac{C_r(i)}{\mathrm{T}_r(i)}. \quad (2)$$

Where $C_r(i)$ and $T_r(i)$ can be calculated with the recursive function as described in Eq. (3).

Table 1
The parameters and their physical significance in the DMPTCP transfer model.

Parameters	Physical Significance
BW_r	The bandwidth of subflow r
$CW_r(i)$	The congestion window of subflow r during the ith round
$SST_r(i)$	The slow start threshold of subflow r during the ith round
$RTT_r(i)$	The RTT of subflow r during the ith round
$P_r(i)$	The packet loss rate of subflow r during the ith round
$C_r(i)$	The accumulated number of successful segements transmissions until the ith round
$T_r(i)$	The accumulated data transmission time until the ith round.
b	The number of segements that are acknowledged by a received ACK.
ck_size	The segements size.
sk_size	The ACK size.

$$\begin{cases} C_r(i) = C_r(i-1) + CW_r(i) - \sum_{l=0}^{CW_r(i)} P(CW_r(i), l) * l, \\ T_r(i) = T_r(i-1) + RTT_r(i) + \dfrac{CW_r(i) * ck\_size + CW_r(i)/b * sk\_size}{BW_r}. \end{cases} \tag{3}$$

In Eq. (3), $C_r(i-1)$ and $CW_r(i)$ denote the accumulated data amount successfully transmitted until the $(i-1)^{th}$ round and the congestion window size of subflow r at the ith round, respectively. The probability of multiple packet loss is calculated with a binomial distribution as shown in Eq. (4):

$$\mathrm{P}(w, r) = \binom{r}{w} * p^r * (1-p)^{w-r}. \tag{4}$$

Thus, the number of lost packets during this transmission can be expressed as $\sum_{l=0}^{CW_r(i)} P(CW_r(i), l) * l$, where l is the corresponding packet loss rate.

In the calculation of $T_r(i)$ according to Eq. (3), $T_r(i-1)$ and $RTT_r(i)$ represent the accumulated transmission time until the $(i-1)^{th}$ round and the smoothed round trip time of subflow r at the ith round, respectively. $\frac{CW_r(i)*ck\_size+CW_r(i)/b*sk\_size}{BW_r}$ denotes the transmission time for $CW_r(i)$ data packets and the corresponding acknowledgments, where b is the number of chunks that are acknowledged by a received ACK. Similar to regular TCP, in MPTCP, the receiver sends one cumulative ACK (or SACK) for every two consecutive packets received, so b is typically 2.

In this recursive function shown in Eq. (3), the transitions of $CW_r(i)$ are analyzed based on the congestion control behavior of MPTCP over each subflow. We consider the following four cases: slow start, congestion avoidance, fast recovery and time out, that control the change of the congestion window size. Given the state of the $(i-1)^{th}$ round, the arrays $CW_r(i-1)$, $SST_r(i-1)$, $P_r(i-1)$, the transmission procedure of the ith round can be analyzed as follows.

Let $p_r^0(i)$ denote the possibility that all the packets were successfully transmitted. $p_r^0(i)$ can be described as:

$$p_r^0(i) = (1-p)^{CW_r(i-1)}. \tag{5}$$

In this situation, the congestion window can be transferred according to Eq. (6) and SSThresh remains unchanged as shown in Eq. (7).

$$CW_r^0(i) = \begin{cases} CW_r(i-1) + 1 & if\ CW_r(i-1) \geq \mathrm{SST}_{r(i-1)}, \\ 2 * CW_r(i-1) & if\ CW_r(i-1) < \mathrm{SST}_{r(i-1)}. \end{cases} \tag{6}$$

$$\mathrm{SST}_r^0(i) = \mathrm{SST}_r(i-1). \tag{7}$$

$$P_r^1(i) = \sum_{l=CW_r(i-1)-3}^{CW_r(i-1)} P(CW_r(i-1), l). \tag{8}$$

Next, the TCP time out occurs when the sender cannot receive three duplicate ACKs, and thus forces the corresponding connection to wait for an RTO as shown in Eq. (8). When this occurs, half of the current congestion window size is saved as SSThresh and slow start begins again from its initial congestion window size as shown in Eqs. (9) and (10).

$$CW_r^1(i) = CW_r(0). \tag{9}$$

$$\mathrm{SST}_r^1(i) = \frac{CW_r(i-1)}{2}. \tag{10}$$

Finally, when packet loss occurs and the TCP sender receives a specified number of identical acknowledgements which is usually set to three (that is, a total of four acknowledgements with the same acknowledgement number) as depicted in Eq. (11), the sender can be reasonably confident that the segment with the next higher sequence number is dropped, and will not arrive out of order. The sender will then update the SSThresh and the congestion window as shown in Eqs. (13) and (12), respectively, and retransmit the missing segment.

$$P_r^2(i) = \sum_{l=1}^{CW_r(i-1)-4} P(CW_r(i-1), l). \tag{11}$$

$$CW_r^2(i) = \frac{CW_r(i-1)}{2}. \tag{12}$$

$$\mathrm{SSThresh}_r^2(i) = \frac{CW_r(i-1)}{2}. \tag{13}$$

Based on the above analysis, the number of successful data transmissions in this $(i)^{th}$ round, denoted as $C_r(i)$, is given by the sum of the product of the possibilities and the corresponding congestion window sizes for all conditions,

$$C_r(i) = \sum_{m=0}^{2} P_r^m(i) CW_r^m(i). \tag{14}$$

To fully explain this model, we use a simple example presented in Fig. 7. The figure illustrates the transitions of the congestion window size, the slow-start threshold and the successful transmission probabilities for every round of subflow r. In the example, we assume that the initial value of threshold is 80 packets and the initial congestion window is 10.

For the first round, subflow r is in the slow start phase and three conditions exist: (1) All packets are successfully transmitted, the congestion window size is doubled (node (20,80)). (2) Packets loss occurs and subflow r enters fast recovery (node (5,5)). (3) Timeout happens (node (1,5)). Then, during the second round, similarly, each node has three children and there are 9 nodes. Each transition probability is computed based on Eqs. (5), (8) and (11). The transferred data amount in a certain round can be obtained according to Eq. (14). Then, we can get the total successful transmissions with Eq. (3).

In this way, given initial congestion window size $CW_r(0)$, the packet loss rate p_r and the slow start threshold, by applying the above algorithm iteratively, we are able to calculate the link throughput until the specified round is reached.

To alleviate the computation overhead, in our implementation, we merge the nodes in the same round which have the same congestion window size and slow start threshold by summing the probability of the corresponding nodes as in (Ma et al., 2007).

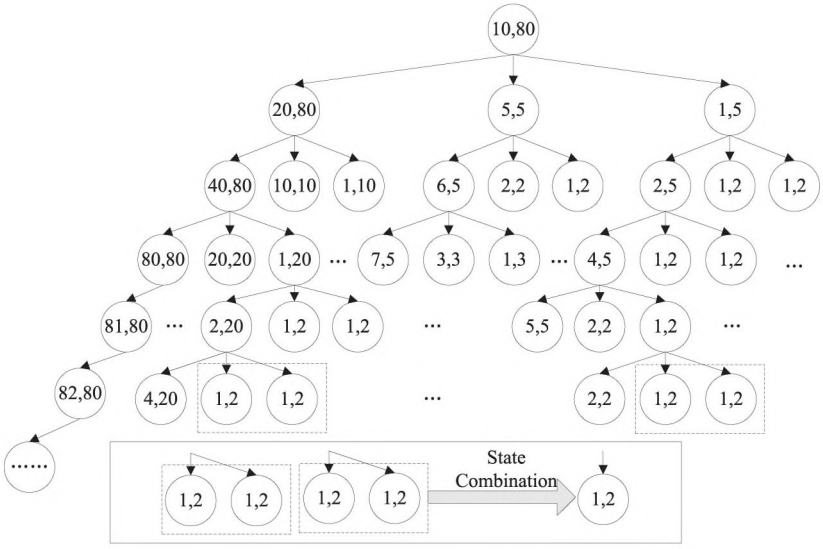


Fig. 7. The illustration of the proposed analytical model for certain subflow r in each RTT round.

4.2. The proposed DMPTCP algorithm

We now start to present the proposed DMPTCP algorithm. We first give the basic idea of DMPTCP and explain how the subflows are selected. We then use an example to present the detail description.

4.2.1. The basic idea of DMPTCP

As shown in Algorithm 1, DMPTCP employs the transmission model of MPTCP to select the corresponding subflows to improve the performance of both short flows and long flows.

We first sort all the available subflows in the ascending order based on the round-trip time of each path i, RTT_i. Then the amount of rounds being transferred on the better paths $(1, 2 \ldots i-1)$ compared to path i is calculated based on the ratio between path i and other paths j (where $j < i$) as shown in Algorithm 1 (Line 7). Specifically, for each path j (where $j < i$), the RTT ratio $\eta_j = RTT_i/RTT_j$ between path i and path j determines the number of TCP rounds that path j can complete before path i does if the two paths start the transmissions simultaneously. Then, the amount of data, denoted as $C_j(\eta)$, which path j can carry during the η rounds can be predicted according to the MPTCP transmission model proposed in Section 4.1. Finally, the total amount of data C_i that can be transferred before path i finishes the first round-trip round can be calculated as shown in Eq. (15) (Lines 8–9 of Algorithm 1).

$$C_i = \sum_{1 \le j < i} C_j(RTT_i/RTT_j) \tag{15}$$

Based on the estimated data amount, DMPTCP selects the suitable set of subflows for corresponding applications. The motivation behind the DMPTCP algorithm is that if the transmission of the given application can be completed within RTT_i by other subflows whose RTT is lower than path i, the first $(i-1)$ paths are selected and subflow i will not be used. To this end, we compare the estimated data amount C_i with the filesize. If $C_i <$ *filesize*, one more subflow, namely, subflow i, will be also taken into account. This process continues until $C_i \ge$ *filesize*. Then, the subflows $s_1, s_2, .s_j \ldots . s_{i-1}$ are selected for this application as shown in Lines 10–13 in Algorithm 1.

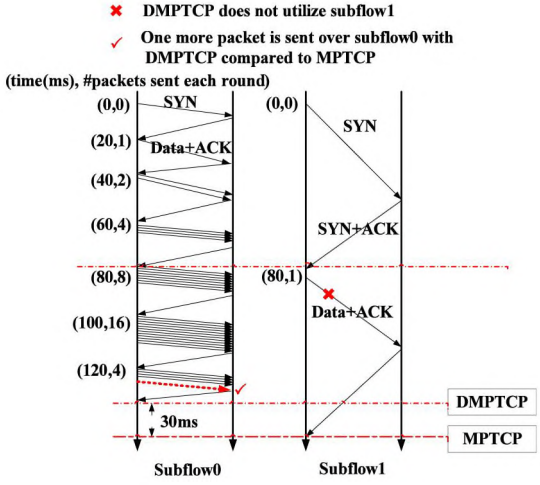


Fig. 8. The workflow of DMPTCP and MPTCP where the RTT of the subflows are 20 ms and 80 ms. MPTCP utilizes both subflows while DMPTCP only select subflow0 and thus reduces its completion time about 30 ms.

Considering that the network environment is constantly changing, the selected path are updated every RTT round.

4.2.2. Detailed description

In this subsection, we illustrate the DMPTCP algorithm in detail with a simple example. Considering the scenario where a 50 KB application transferred across two disjoint subflows. The bandwidth is 100 Mbps and the RTT of subflow0 and subflow1 are set as 20 ms and 80 ms, respectively. The workflow of the proposed DMPTCP algorithm and MPTCP is illustrated in Fig. 8 and analyzed as follows.

In MPTCP, each subflow is a regular TCP connection and is initialized through a three-way handshake at the start time simultaneously. As shown in Fig. 8, subflow0 starts to send data at 40 ms as the last ACK of the three-way handshake is piggybacked on the first data packet. Similarly, subflow1 is established and starts to transfer the first packet at 80 ms. Meanwhile, subflow0 is at the beginning of the 4^{th} round with the congestion window size of 8.

Algorithm 1 The Proposed DMPTCP Algorithm.

Input : A set of sorted paths $P = p_1, p_2, ..., p_n$ and the application filesize fs.

Output: A set of paths $S = s_1, s_2, .s_j.., s_i$ $i(i \in [1, n])$ that selected for certain application.

1 **Initialization at** $t = 0$
2 $k = 0$;
3 **for** *each* $C_i \in C$ **do**
4 $C_i = 0$;
5 **for** *each* $p_i \in P$ $(i \in [2, n])$ **do**
6 **for** $j = 1$ *to* i **do**
7 $\eta_j = RTT_i/RTT_j$;
8 Calculate $C_j(\eta_j)$ according to Eq. (3);
9 $C_i+ = C_j(\eta_j)$;
10 **if** $C_i \geq fs$ **then**
11 put $s_1, s_2, .s_j.., s_{i-1}$ in the set S;
12 return S;

Then, 8 packets and 1 packet are spread across subflow0 and subflow1 respectively with MPTCP. For the proposed DMPTCP, it first measures the current RTT of each subflow and finds that the RTT of subflow1 is 4 times larger than that of subflow0. This indicates that subflow0 can complete 4 rounds before subflow1 finishes the first round. Then, the data amount that can be transferred of subflow0 during the four rounds is estimated according to Eq. (3) with the value of 164K. The application has been transferred about 10 KB with subflow0 before subflow1 is established and 40 KB is waiting for transmission. As the estimated data amount (164K) is larger than the 40K, then only subflow0 is utilized and no packet will be sent with subflow1 according the Algorithm 1 (Line 7).

Finally, DMPTCP finishes the transmission at 121.889 ms. MPTCP finishes its transfer at 121.772 ms over Subflow0. However, the whole MPTCP connection waits for the ACK of subflow1 until it arrives at 151.459 ms.

4.3. *The validation of the analytical model*

The proposed analytical model is implemented in Matlab. To validate the accuracy of the proposed analysis model, we run the simulations and compare the simulation results with the analytical prediction calculated using Matlab. The test scenarios are changed by setting different network propagation delays and random packet loss rates. During the test, the bandwidth is fixed at 1 Mbps.

Fig. 9(a) and (b) show the comparisons of the analytical and simulation results for data transmissions when there are no random packet loss rate. In Fig. 9(a), we show the throughput from both the analysis and the simulations over different transfer sizes from 20 KB to 10 MB when the RTT is 20 ms. Results for different end-to-end delay are shown in Fig. 9(b).

In these two figures, we see that the data produced by the analytical model are closely matched to the simulation results as all kinds of transitions are taken into consideration in the analytical model, making it provide a fairly accurate prediction of each round as shown in Fig. 10 which depicts the change of the throughput over time.

Next, we validate the analytical model for MPTCP data transmission subject to random loss. We employ a random loss error model to simulate error loss on the links. The random packet loss rates are set to be 0.01%, 0.02%, 0.03%, 0.05%, 0.08% and 0.1%. The RTT is 100 ms. We compare the throughput performance obtained numerically using the proposed analytical model with that obtained by simulations. The results are depicted in Fig. 11 and reveal that the analytical model can match the simulation results fairly well in these tests.

5. Simulation evaluation

In this section, we explore the performance revenue by performing the extensive experiments based on ns3.14 under various network conditions. The MPTCP ns3 code is provided by google mptcp group (Google, 2017) For the simulation results, we consider the number of timeouts and retransmits as well as the average flow completion time as the main evaluation metrics.

5.1. *Measurement setting*

We evaluated two scenarios (Ferlin et al., 2016a; Hassayoun et al., 2011), namely, subflows with and without shared links, which are denoted as NL and NSL respectively for the sake of simplicity in the remainder of this article. Fig. 1 illustrates the NSL scenario where M concurrent flows run between the MPTCP client and the MPTCP server with N disjoint subflows. Comparatively, there are common links in the

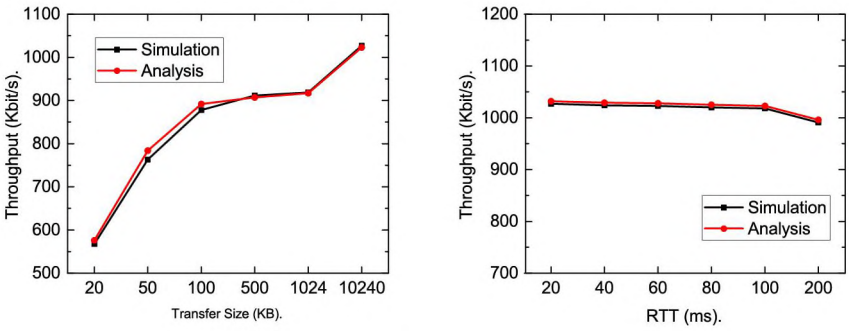


Fig. 9. Comparison of analysis and simulation results.

SL scenario as depicted in Fig. 21, through which all MPTCP subflows are sent.

Following the parameters in (Barrè et al., 2011), the number of subflow N ranges from 1 to 8. For the NSL scenarios, the bandwidth is 100 Mbps and the RTTs of the subflows are set as 10 ms, 20 ms, 30 ms, 40 ms, 60 ms, 80 ms, 90 ms and 100 ms. For the SL scenario, the common link has 100 Mbps capacity and 20 ms RTT (Ferlin et al., 2016a) and the RTTs of each subflow are 20 ms, 30 ms, 40 ms, 50 ms, 60 ms, 70 ms, 80 ms and 90 ms. Unless specifically stated, the default setting of the packet loss rate is 0.

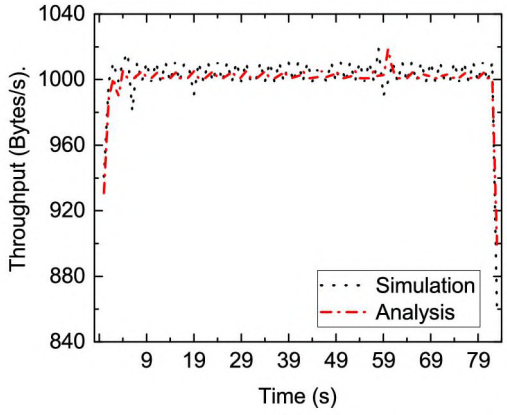


Fig. 10. The instantaneous throughput when the RTT is 20 ms and the transfer size is 10 MB.

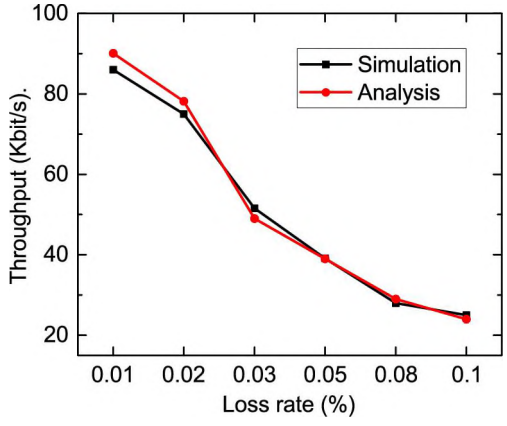


Fig. 11. Comparisons of the analytical and simulation results for data transmissions when there are random packet losses.

The droptail bottleneck queue was set to 1 bandwidth-delay product (BDP), which allows for 100% link utilization with a congestion control that halves its cwnd upon congestion (Ferlin et al., 2016a). The Maximal Segment Size (MSS) in our simulation is set to 1400 bytes (Ni et al., 2015), which is also the packet size at TCP layer. The socket buffer is set following the recommendation adopted in (Ferlin et al., 2016a), i.e., buffer = $\sum_{i=1}^{N} bandwidth_i * RTT_{\max} * 2$, where N is the number of subflows, $bandwidth_i$ is the bandwidth of subflow i and $RTT_{\max}$ is the max RTT over all subflows. The congestion control algorithm adopts RTT-compensation as that in (Ni et al., 2015), which is used widely in MPTCP and provides more friendliness.

Both the web-like cross traffic and the long-lived FTP flows consist of the traffic pattern in our simulation. For the short flows, their transfer size obeys the Pareto distribution with an average file size of 57 KB. This setting is consistent with the real-world Web traffic model in (China Internet Network Information Center, 2017). The size of long-lived TCP flows is set as 15 MB and the number of flows is varied between 1 and 15. Each flow bursts synchronously at 0.1 s. As in (Ferlin et al., 2016a; Kheirkhah et al., 2016), we compared the performance of DMPTCP with regular TCP as well as MPTCP.

Our experiments are divided into three parts. First, we validate whether DMPTCP can avoid the impairments causing suboptimal performance of MPTCP. Then we explain the performance of DMPTCP in a non-shared environment where the subflows have disjoint paths. Finally, we compare the performance of each algorithm when the subflows have common links.

5.2. Impairments analysis

As analyzed in Section 2, MPTCP is inefficient for short flows as the packet reordering caused by the heterogeneous paths and the retransmission timeouts when only a small amount of data is spread over each subflow. In this subsection, we examine whether DMPTCP can alleviate these performance impairments of MPTCP with the topology shown in Fig. 1.

5.2.1. Impact of heterogeneous paths

Fig. 12 depicts the average completion time of a 100 KB application when there are two subflows. The RTT of one subflow is fixed at 10 ms, while the other varies from 10 ms to 50 ms. According to the figure, when the two subflows have the same RTTs, both MPTCP and DMPTCP outperforms TCP as these two algorithms utilize the two subflows simultaneously. However, with the increasing RTT of the second subflow, MPTCP still uses the two established subflows, showing worse performance compared to regular TCP and DMPTCP. Comparably, DMPTCP only use the best path of the two subflows, and thus improving the completion time of short flows compared to MPTCP.

5.2.2. Impact of retransmission timeouts

Fig. 13 shows the completion time of each algorithm when there are four subflows and the RTT of each subflow is 10 ms, 50 ms, 100 ms

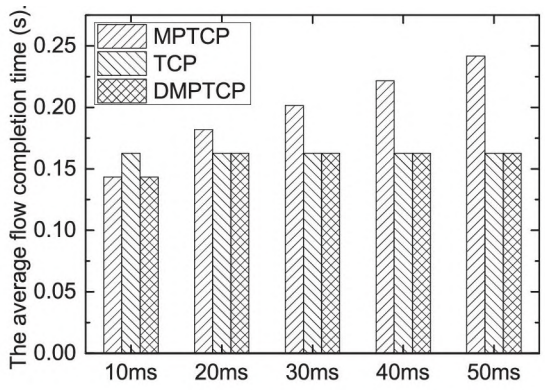


Fig. 12. The completion time of each algorithm transferred across two subflow when the application size is 100 KB.

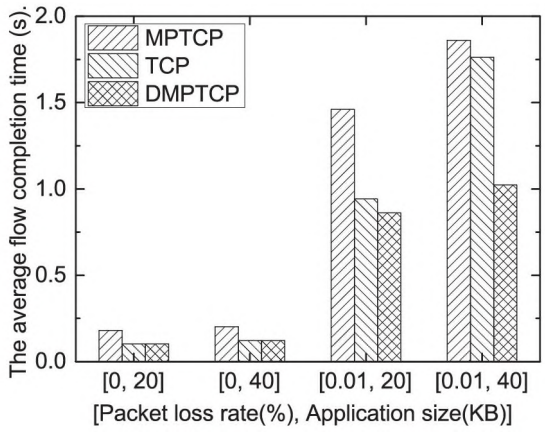


Fig. 13. The completion time of each algorithm when there are four subflows and the RTT of each subflow is 10 ms, 50 ms, 100 ms and 200 ms.

and 200 ms. The application size is 20 KB or 40 KB. As described in the figure, when the random loss rate is 0, DMPTCP obtains the same performance as TCP because it only utilizes the subflow with the lowest RTT. Both DMPTCP and TCP outperform MPTCP as MPTCP utilizes all the established four subflows, causing increased delays.

Besides, it also follows the figure that when there are random packet loss events, DMPTCP outperforms both MPTCP and TCP. To further analyze the reason, we conduct statistics on the retransmission timeout. Take the application of 40 KB file size as an example. The results are shown in Table 2. As depicted in this table, due to the random packet loss, all the three algorithms suffer RTOs. The number of RTOs is up to 10 with MPTCP which is significantly higher than that of DMPTCP and TCP, resulting the longest flow completion time.

Table 2
The time and subflows that the retransmission timeout occurs with each algorithm.

MPTCP		TCP		DMPTCP	
SubflowID	Time (s)	SubflowID	Time	SubflowID	Time
1	0.240266	1	0.410797	1	0.240266
1	1.06126	1	0.640896	3	0.460507
1	1.26126	1	1.29289	3	0.660747
3	0.460507	1	1.73392		
3	0.660747				
4	0.660747				
4	0.860747				
4	1.06099				
4	1.26099				
4	1.46099				

In addition, as shown in Fig. 13, DMPTCP also outperforms TCP when there is random packet loss. The analysis shows that this benefits from additional paths that DMPTCP utilized compared to regular TCP and the reduced RTOs shown in Table 2.

Based on the above analysis, we can conclude that by selecting the appropriate set of subflows for each application, DMPTCP can improve the flow completion time compared to MPTCP benefit from the reduced timeout as well as decreased delay caused by heterogeneous path characteristics. In the following, more experiments are conducted to evaluate the performance of DMPTCP in various network conditions.

5.3. Non-shared links

In this subsection, we first conduct the experiments with concurrent short flows to further validate the effectiveness of DMPTCP. Then, the performance of all the three algorithms with long-lived flows is evaluated to verify whether DMPTCP maintains the high performance for long flows as MPTCP. The topology is shown in Fig. 1.

Fig. 14 shows the experimental results with varying number of subflows of both MPTCP and DMPTCP. According to this figure, for the short web-like flows, MPTCP performs worse with increasing number of established subflows. Both network scenarios show similar behavior. Thus, in the following of the paper, we conduct experiments in the default network scenario where the bandwidth is 100 Mbps and the RTT ranges from 10 ms to 100 ms according to the set in prior work (Barré et al., 2011).

To analyze the reason we conduct experiments to investigate the number of subflows utilized with different flow sizes without packet loss. The results are depicted in Table 3 when the bandwidth is 100 Mbps and there are 8 subflows established. In our experiments, the flows ranges from 19 KB to 87 KB and have an average size of 57 KB. According to Table 3, MPTCP utilizes about 2–5 subflows for each application while DMPTCP only spreads packets across two subflows with smaller RTTs according to Algorithm 1. Consequently, MPTCP has the same completion time with DMPTCP when only one or two subflows are established, and brings about increased delays with increasing number of subflows.

Further, we do experiments to investigate the performance of each algorithm in lossy network conditions. The experimental results are depicted in Fig. 15, which again prove that MPTCP is not efficient for short flows compared to regular TCP, and the performance becomes even worse with the increasing loss rate. The reason lies in that MPTCP spreads data across all available subflows such that the data amount as well as the congestion window over each subflow is quite small over its lifetime. Consequently, the packet loss cannot be recovered from the fast retransmit and RTO happens. These can be clearly validated in Fig. 16 which illustrates the transmitted sequence numbers of an application whose size is 21685B with MPTCP over sublfow0, sublfow2, sublfow3 and sublfow4. As depicted in this figure, the data amount assigned to each of these subflows is only one or two packets. What's worse is that for the other subflows, i.e., subflow1, subflow5, subflow6 and subflow7, they do not successfully transmit one packet as shown in Fig. 17 until a packet loss event occurs.

On the contrary, DMPTCP can limit the number of subflows used for short flows. For the above application with the size of 21685B, the data are only transferred over subflow0 and subflow1 with DMPTCP. Thus, the data amount transferred over each subflow increases a lot as shown in Fig. 18, and therefore the RTOs are significantly alleviated as analyzed in Section 5.2.

Besides, as shown in Fig. 15, DMPTCP outperforms TCP. As analyzed before, the reason lies in that DMPTCP selects the appropriate subflow set for simultaneously transmission while TCP only utilizes one path for all applications whose sequence number is illustrated in Fig. 19.

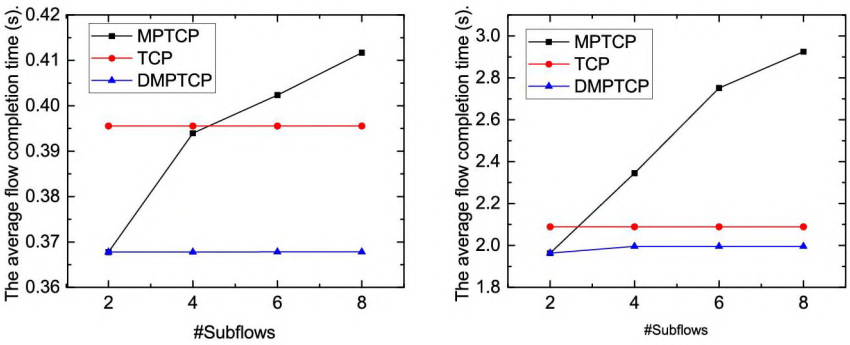


Fig. 14. The average completion time of 30 short web-like flows with varying established #Subflows. (a) The bandwidth is 100 Mbps and the RTTs of the subflows are 10 ms, 20 ms, 30 ms, 40 ms, 60 ms, 80 ms, 90 ms and 100 ms. (b) The bandwidth is 10 Mbps and the RTTs of the subflows are 100 ms, 100 ms, 200 ms, 200 ms, 400 ms, 400 ms, 500 ms and 500 ms.

Table 3
The number of subflows utilized by MPTCP and DMPTCP with different flow sizes.

Sizes	MPTCP	DMPTCP	Sizes	MPTCP	DMPTCP
10K	1	1	600K	8	2
20K	2	2	800K	8	2
50K	4	2	1M	8	2
100K	5	2	2M	8	3
200K	7	2	5M	8	4
400K	8	2	10M	8	8

Finally, experiments with long flows are done to validate whether DMPTCP can maintain the high performance for long flows as MPTCP does. The results are shown in Fig. 20. As expected, DMPTCP has the similar behavior as MPTCP as both algorithms utilizes all established subflows. Besides, as depicted in the figure, compared to TCP, MPTCP as well as DMPTCP can improve the completion time of long flows benefit from the multiple concurrent subflows, which is in accordance with existing previous works (Dong et al., 2016; Kheirkhah et al., 2016; Yedugundla et al., 2016).

5.4. Shared links

In this subsection, the performance of each algorithm when the subflows have common links (i.e., the SL scenario) is evaluated with the topology shown in Fig. 21.

Fig. 22 shows the average flow completion time with varying loss rates. According to this figure, for short web-like flows, the shared scenarios have similar results as that in the non-shared ones. Specifically, DMPTCP outperforms MPTCP as well as TCP, benefiting from selecting the corresponding set of subflows for each application. As a result,

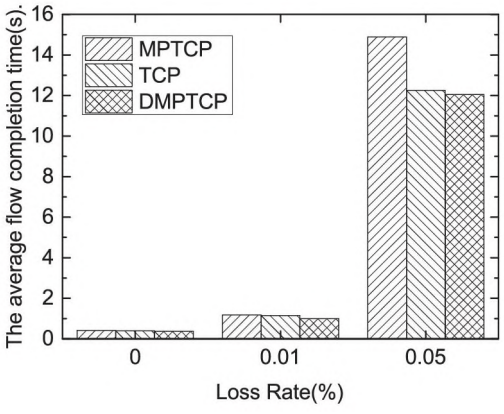


Fig. 15. The average completion time of short web-like flows with varying packet loss rate.

DMPTCP can utilize multiple paths simultaneously to improve the flow completion time while alleviate the frequent RTOs in traditional MPTCP which uses all available paths.

However, the difference also exists. According to Figs. 22 and 15, compared to the disjoint path situation, the performance gain of DMPTCP compared to MPTCP is not so significant in the shared link scenario. A detailed analysis shows that in all of the current existing MPTCP's coupled congestion control algorithms, MPTCP subflows in slow-start are uncoupled, i.e., each behaving as regular TCP and increasing exponentially (Barik et al., 2016). Therefore, MPTCP can be more aggressive compared to a concurrent TCP flow at the shared bottleneck in the slow start phase. In our experiments, the short flows have an average size of 57 Kb, and thus, for these short flows, the slow start behavior becomes of critical importance for the performance.

This can be validated in Fig. 22 when the packet loss rate is 0. As depicted in this figure, MPTCP outperforms both DMPTCP and TCP because of its aggressive behavior in slow start phase. However, this phenomenon terminates when the packet loss exists and the performance of MPTCP is getting worse and worse with the increasing loss rate, compared to DMPTCP as well as TCP. This is due to the RTOs caused by the small congestion window size over each subflow because the data is spread across all available subflows in MPTCP. Meanwhile, benefiting from the large number of subflows utilized compare to DMPTCP, MPTCP performs better compared to that in the non-shared scenario.

The experimental results of long flows are shown in Fig. 23. As depicted in this figure, DMPTCP and MPTCP also have the same throughout gain as in the non-shared scenario because both the two algorithms utilizes all the established subflows for long-lived flows in our experiments when the flow size is 15 MB as depicted in Table 3.

We can also obtain from this figure that when the number of the competing flows is large (i.e. 15), all the three algorithms show nearly the same average flow completion time. As for long-lived flows, the congestion avoidance phase plays the dominant role for the performance. In the existing MPTCP congestion control protocols, the congestion avoidance phase of all the subflows is coupled to keep bottleneck fairness with regular single-path TCP. This is also one of the three design goals of the MPTCP congestion control (Dong et al., 2016; Wischik et al., 2011). As a result, all the three algorithms show similar performance in the shared links scenario with long flows.

However, small competing flows (e.g., 2 and 4 long-lived flows) show interesting results. Both MPTCP and DMPTCP outperform TCP. A detailed analysis shows that this is because when the number of the concurrent flows is small, the shared links are not congested and MPTCP can benefit from concurrent multiple subflows. However, with the increasing competing flows, the common link becomes the bottleneck and constraints the performance of MPTCP.

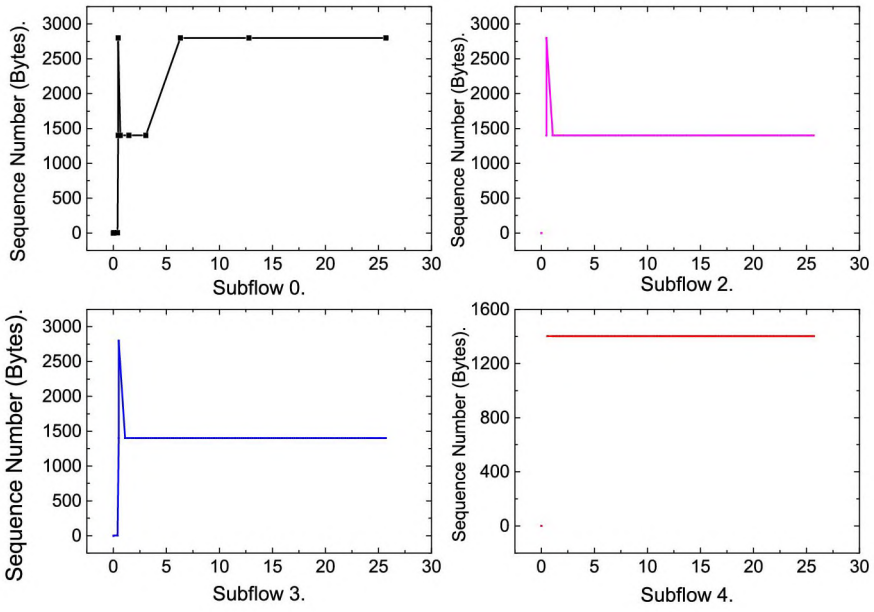
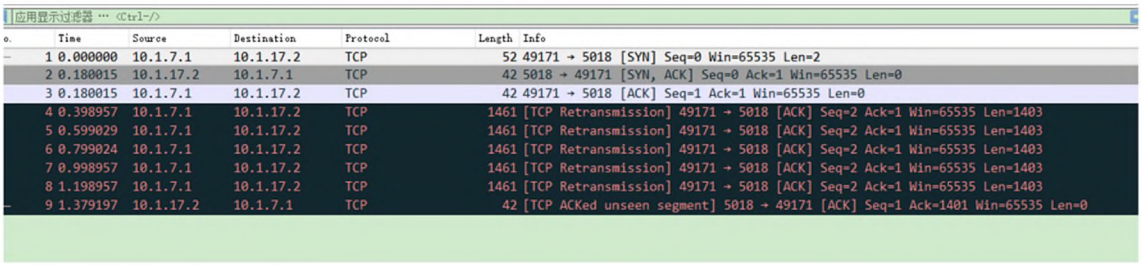


Fig. 16. The MPTCP's steven sequence number analyzed with wireshark.



o.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	10.1.7.1	10.1.17.2	TCP	52	49171 → 5018 [SYN] Seq=0 Win=65535 Len=2
2	0.180015	10.1.17.2	10.1.7.1	TCP	42	5018 → 49171 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
3	0.180015	10.1.7.1	10.1.17.2	TCP	42	49171 → 5018 [ACK] Seq=1 Ack=1 Win=65535 Len=0
4	0.398957	10.1.7.1	10.1.17.2	TCP	1461	[TCP Retransmission] 49171 → 5018 [ACK] Seq=2 Ack=1 Win=65535 Len=1403
5	0.599029	10.1.7.1	10.1.17.2	TCP	1461	[TCP Retransmission] 49171 → 5018 [ACK] Seq=2 Ack=1 Win=65535 Len=1403
6	0.799024	10.1.7.1	10.1.17.2	TCP	1461	[TCP Retransmission] 49171 → 5018 [ACK] Seq=2 Ack=1 Win=65535 Len=1403
7	0.998957	10.1.7.1	10.1.17.2	TCP	1461	[TCP Retransmission] 49171 → 5018 [ACK] Seq=2 Ack=1 Win=65535 Len=1403
8	1.198957	10.1.7.1	10.1.17.2	TCP	1461	[TCP Retransmission] 49171 → 5018 [ACK] Seq=2 Ack=1 Win=65535 Len=1403
9	1.379197	10.1.17.2	10.1.7.1	TCP	42	[TCP ACKed unseen segment] 5018 → 49171 [ACK] Seq=1 Ack=1401 Win=65535 Len=0

Fig. 17. The transmission process of MPTCP over subflow1, subflow5, subflow6 and subflow7.

5.5. *Mixing of flows*

We further conduct experiments to investigate the behavior of MPTCP when competing with regular TCP flows in this subsection. The network topology for these experiments is shown in Fig. 24, where both MPTCP flows and regular TCP flows compete for the common link. The short flows as well the long-lived flows are taken into consideration.

Fig. 25(a) shows the completion time of short flows in a setting that 30 eight-path MPTCP flows and 30 regular single-path TCP flows competes for the common link. As depicted in the figure, both DMPTCP and MPTCP are more aggressive compared to regular TCP, and thus achieve better performance. As analyzed in Section 5.4, it is because that the slow start of each subflow in MPTCP behaves like regular single-path TCP and increases exponentially every RTT. In the simulation, each MPTCP consists of 8 subflows, making MPTCP more aggressive to complete with TCP flows for the short web-like flows where the slow start plays an important role for the performance.

The figure also reveals that TCP has a better performance while competing with DMPTCP than that while competing with MPTCP. The reason is that DMPTCP uses the less number of subflows than MPTCP.

Fig. 25(b) shows the average completion time of long-lived flows when 10 multipath flows competes with 10 regular TCP flows. As depicted in this figure, DMPTCP and MPTCP shows a similar behavior for long flows which is consistent with the results described above. Besides, as expected, both MPTCP and DMPTCP can fairly share the

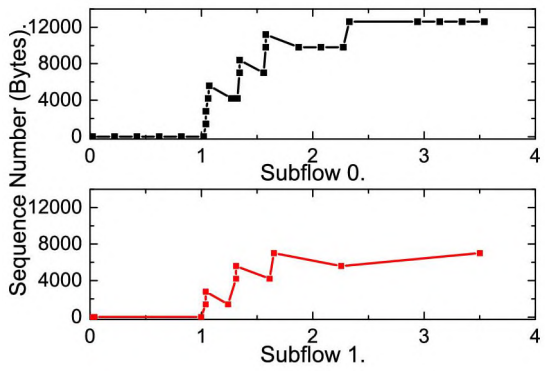


Fig. 18. The DMPTCP's steven sequence number analyzed with wireshark.

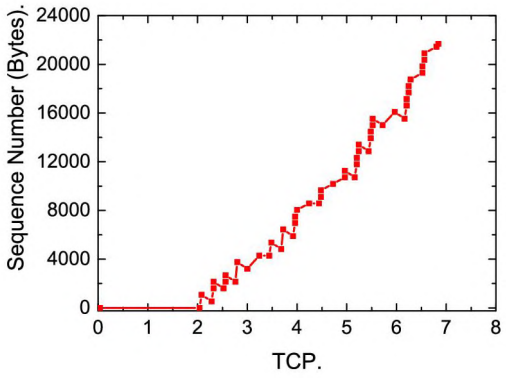


Fig. 19. The TCP's steven sequence number analyzed with wireshark.

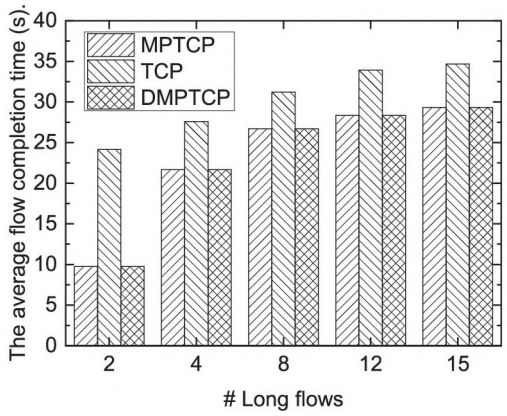


Fig. 20. The average completion time of long flows when the #Subflow is 8.

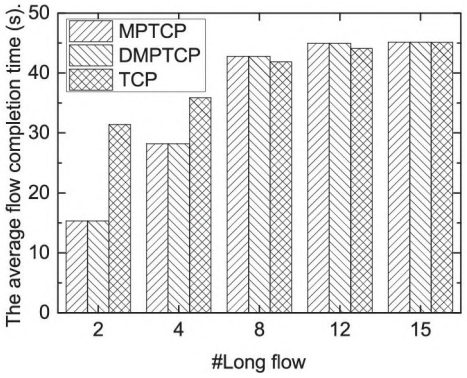


Fig. 23. The average completion time of long flows in SL scenario with topology shown in Fig. 21.

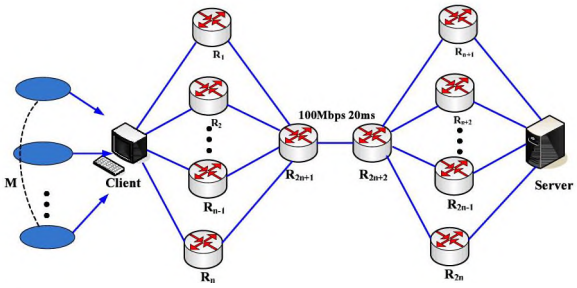


Fig. 21. The topology of the SL scenario where the subflows have common links.

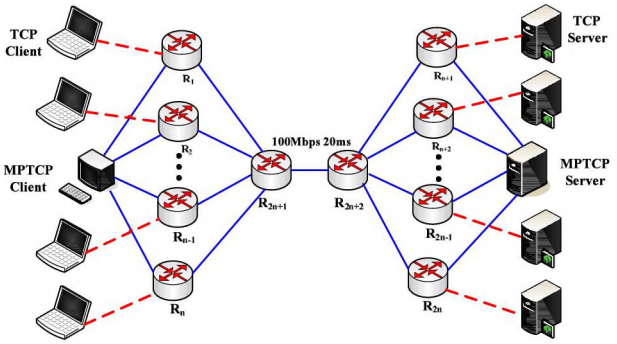


Fig. 24. Topology for experiments when multipath TCP flows and regular single-path TCP flows competes for the common link.

common links with regular TCP for long flows because of the coupled congestion avoidance with the Jain Fairness Index of 0.999247312.

6. Testbed experiment

In this section, we validate the proposed DMPTCP algorithm by conducting experiments on our testbed and comparing its performance with other of four existing algorithms: BLEST (Ferlin et al., 2016b), DAPS (Kuhn et al., 2014; Sarwar et al., 2013), OTIAS (Yang et al., 2014) and the default minimum RTT First (minRTT) scheduler. We use the publicly available Linux code of BLEST, DAPS and OTIAS linked in BLEST (Ferlin et al., 2016b), and also modified the Linux kernel to implement DMPTCP.

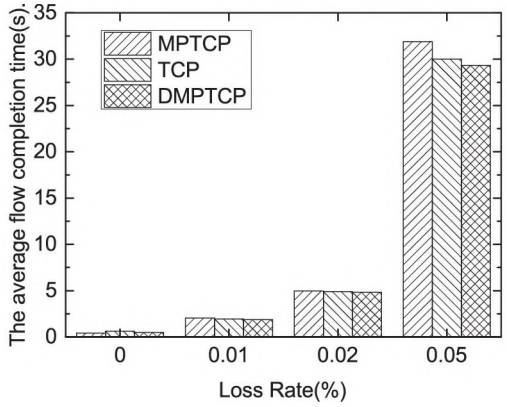
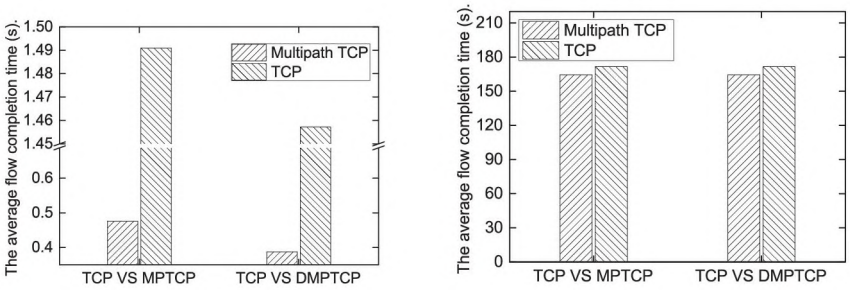


Fig. 22. The average completion time of short web-like flows in SL scenario with topology shown in Fig. 21.

6.1. Testbed construction and experimental methodology

The deployed experiment testbed consists of three file servers, two computers with WANem and three clients, which constitutes the network topology shown in Fig. 26 by means of routing configurations. The topology is widely used in the existing works (Ferlin et al., 2016b; Kuhn et al., 2014). Both the clients and the servers are running Linux ubuntu12.10 OS with kernel version 3.14.33 that has already applied the protocol patches. The servers are running on the Dell T1500, equipped with the Intel Xeon E5620 (2.4 GHz/12M), 16 GB RAM and a 600 GB Hard Disk. The clients are running on the DELL optiplex 745, equipped with Intel PentiumD 3.4G processor, 512 MB RAM and 160 GB hard disk. As shown in Fig. 26, one of the clients labeled S_2 is equipped with two Gigabit network interface cards to establish two subflows between the MPTCP server D_2. We consider this as the common scenario (e.g., a client having two access networks like WiFi/4G) (Barik et al., 2016; Ferlin et al., 2016b). R_1 and R_2 serve as two routers which run WANem to construct a two-way bottleneck link. WANem is a wide area network emulator that supports various wide area network features such as bandwidth limitation, latency, packet loss, network disconnection and so on.

GNU Wget is used to generate TCP data traffic by retrieving binary documents through HTTP. Each data point is obtained by computing the average value of the results from ten rounds of execution. All the experimental data is captured at the clients using tcpdump and then is analyzed with wireshark. The binary documents range from 16 KB to 8 MB. Meanwhile, a wide range of network environments are considered: the round-trip time is in the range from 20 ms to 400 ms, packet



(a) Short flows. (b) Long flows.

Fig. 25. The average completion time of each algorithm with topology shown in Fig. 24 when multipath flows compete with regular single-path TCP flows.

loss rate is changed from 0.1% to 5% and the bottleneck bandwidth also varies from 2 Mbps to 100 Mbps.

In the experiments, we compare the performance of each algorithm in terms of Goodput as well as the flow completion time. The retransmissions as well as the amount of data spread over each subflow (the contribution of each subflow) is also taken into consideration for further analysis.

6.2. *Experimental results*

In this section, we firstly study the performance of each algorithm with various network configurations to illustrate how each scheduler solves the challenges of MPTCP in heterogeneous scenarios. Then, the Goodput gain and the flow completion time of each algorithm in WLAN-3G scenarios are investigated.

6.2.1. *Challenges of MPTCP and countermeasures of each scheduler*

In this subsection, we conduct experiments of each algorithm with topology shown in Fig. 26 when the flow size ranges from 16 KB to 8 MB in various network conditions.

Fig. 27 illustrates the flow completion time of each algorithm with varying RTTs when the bandwidth is 8 Mbps. As depicted in this figure, when the application size is 16K, all the five algorithms show similar performance. The reason is as follows. Multipath TCP starts the connection by establishing an initial TCP subflow with a standard TCP 3-way handshake. If the peer host supports Multipath TCP, it will advertise all additional IP addresses to the connection initiator during this procedure (Barik et al., 2016). Then, the additional subflows are able to join in the Multipath TCP connection. That's, the additional subflow is established after the 3-way handshake of the initial sublfow. However, the initial congestion window is 10 in the current Linux implementation. This makes sure that the application with size of 16 KB can be successfully transferred before the establishment of the second subflow. Thus, all the five algorithms present similar flow completion time as regular TCP with the value of about 0.216s.

Meanwhile, the flow completion time of each algorithm differs when the application size is 256 KB especially when the RTT of the second subflow is 100 ms, where the default minRTT scheduler finishes the transmission much slower than other algorithms. During our analysis, we find that DMPTCP, BLEST, OTIAS as well as DAPS only transferred over the fast subflow in this situation. However, minRTT spreads data across both the two subflows as the congestion window of the fast subflow is full and not available. The statistics reveals that the second subflow only transfers one packet of 1500 bytes (containing the TCP header). This results in that the default scheduler competes the transmission about 230 ms later than other algorithms.

Then, we investigate the behavior of each algorithm when random packet loss exists in the same network scenario described above, i.e., the bandwidth is 8 Mbps and the RTT of subflow0 and subflow1 is 20 ms and 100 ms, respectively. The results are illustrated in Fig. 28. According to this figure, when the packet loss rate is low, the difference between each algorithm is small. However, DAPS as well as the default scheduler performs worse with the increasing loss rate. To find the reasons, we analyze the traced PCAP file with wireshark. Take the scenario when the packet loss rate is 3% as an example. The retransmission ratio of BLEST, DAPS, minRTT, OTIAS and DMPTCP is 2.04%, 7.04%, 4.17%, 2.14% and 2.03%, respectively. The reason of the higher

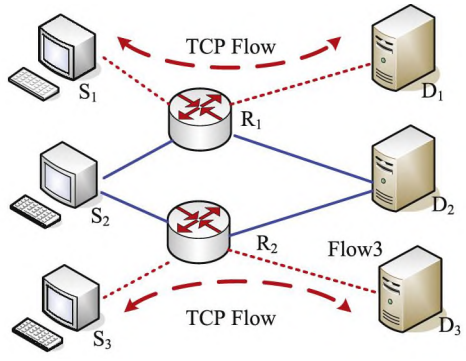


Fig. 26. TestBed topology.

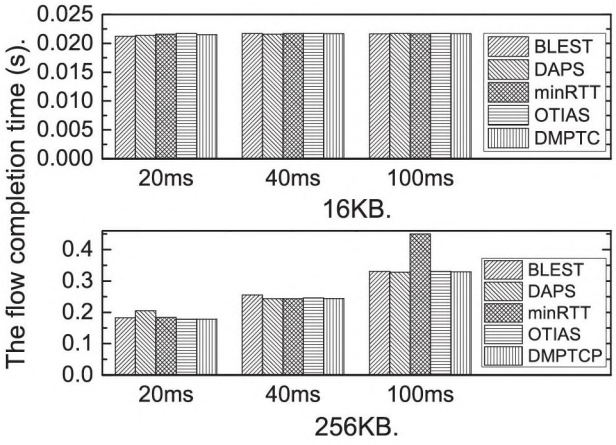


Fig. 27. The flow completion time when the flow size is 16 KB and 256 KB where the bandwidth is 8 Mbps and the RTT of one flow is fixed at 20 ms while that of the other subflow varies from 20 ms to 100 ms.

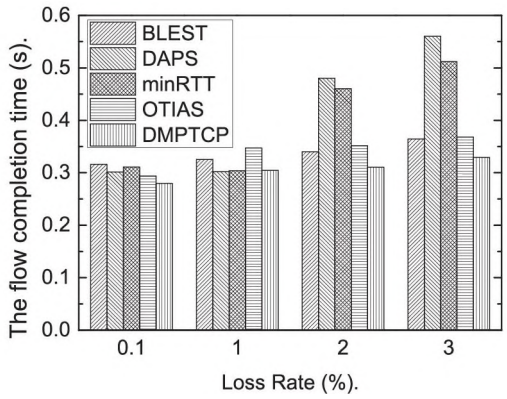


Fig. 28. The flow completion time of a 256 KB application when random packet loss exits where the bandwidth is 8 Mbps and the RTT of two subflows is 20 ms and 100 ms.

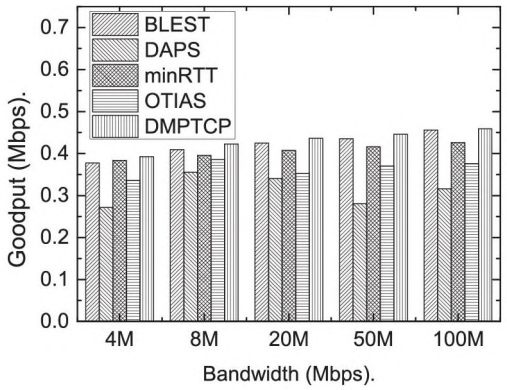


Fig. 29. The Goodput of the 8 MB application with varying bandwidth where the random packet loss rate is 3% and the RTT of two subflows is 20 ms and 100 ms.

Table 4
The number of packets transferred over each subflow with different algorithms corresponding to the results shown in Fig. 28.

Algorithm	Subflow0	Subflow1
BLEST	163	33
DAPS	176	23
minRTT	159	33
OTIAS	186	11
DMPTCP	157	30

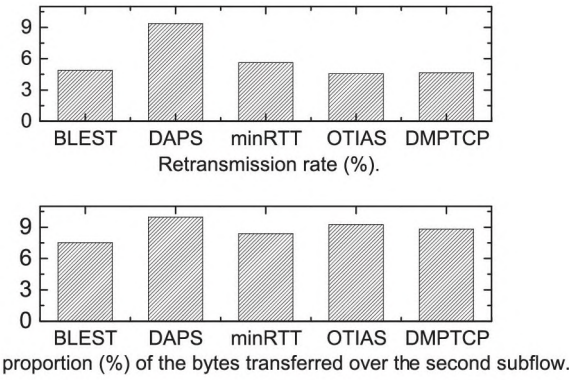


Fig. 30. The retransmission rate and the proportion of the bytes transferred over the second subflow of each algorithm illustrated in Fig. 29.

retransmission ratio of DAPS lies in that the algorithm cannot select subflows at run time and cannot react quickly enough to changes on the paths which can only adjust the packet transfer in the next scheduling run as declared in (Ferlin et al., 2016b). This condition becomes even worse when packet loss occurs as DAPS cannot employ the retransmission mechanism, retransmitting a packet on the fastest subflow.

In addition, we can also obtain from this figure that DMPTCP outperforms BLEST. Although both DMPTCP and BLEST aim to improve the performance of MPTCP, BLEST decides which subflow to use based on the estimation whether a path will cause send-window blocking rather than the information of the application. As a result, as depicted in Table 4, the number of packets transferred over the slow path with BLEST and DMPTCP are 33 packets and 30 packets, respectively in this condition. As a result, DMPTCP can complete the transmission within two rounds while BLEST needs three rounds. In addition, BLEST only takes the congestion avoidance phase into consideration when estimating the amount of data that decides whether to use the slower subflow or not. The estimation is inaccurate for short flows as slow start is of critical importance for these flows, causing a higher flow completion time.

Finally, the performance comparison of each algorithm with long flows (8 MB) is also investigated. The results are depicted in Figs. 29 and 30.

According to Fig. 29, BLEST, minRTT and DMPTCP outperforms DAPS and OTIAS. Compared to the MPTCP's default minRTT scheduler, DAPS provides a Goodput decrease from 10.12% to 32.67% and a retransmission ratio increase up to 66.59% (Fig. 30) because its scheduling round is the LCM (Least Common Multiple) of each subflow's RTT, which is too long to react quickly enough to changes on the paths in this lossy network condition. Besides, OTIAS decides which subflow to use on per-packet basis, and it takes into account the RTTs, the queue sizes, the congestion window of the subflows. Compared to the MPTCP's default scheduler (minRTT), it takes into account more information from the subflows, showing the lowest retransmit rate among all algorithms. However, OTIAS also builds queues on the subflows and cannot react immediately to the network changes, leading to a Goodput decrease in the range from 2.47% to 13.43%.

On the contrary, BLEST, minRTT as well as DMPTCP can dynamically adjust the packet scheduling policy based on the network dynamics, and achieves higher Goodput compared to DAPS and OTIAS. In addition, DMPTCP outperforms BLEST, because BLEST adapts scheduling to prevent the send-window blocking over the fast subflow. Specifically, BLEST estimates the amount of data X that will be sent on the fast path during RTT_S which is the RTT of the slow path. If X is larger than the send window of the fast subflow, the scheduler waits for the faster subflow to become available and will not utilize the slow path. This makes BLEST cannot fully utilize the second subflow in this condition and leads to the proportion of the bytes transferred over the second subflow is lower than other algorithms (Fig. 30).

6.2.2. *WLAN-3G scenarios*

In this subsection, we further conduct experiments when the network parameters are set following the link characteristics of WLAN and 3G links (Ferlin et al., 2016b) as shown in Table 5 with WANem. We conduct experiments with the topology shown in Fig. 26, and three sce-

Table 5
Link parameters.

Link	Bandwidth (Mbps)	Loss rate (%)	RTT (ms)
WLAN	25	1	25
3G	5	0	65

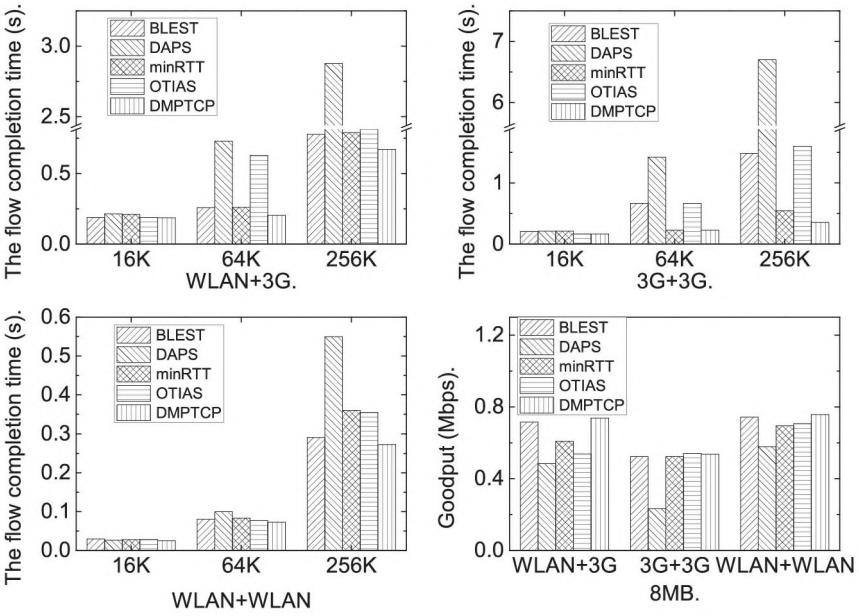


Fig. 31. The experimental results under WLAN-3G network scenarios.

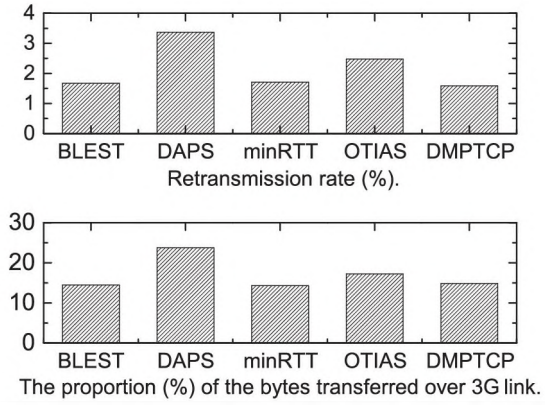


Fig. 32. The retransmission rate and the proportion of the bytes transferred over the 3G link when the flow size is 256 KB in the WLAN+3G scenario.

narios are taken into consideration, namely, WLAN+3G, WLAN+WLAN and 3G+3G. Take the scene of WLAN+3G as an example, which means for an MPTCP connection, one subflow is WLAN link while the other sublfow is 3G link. We use 10 concurrent TCP Reno flows as the background traffic over each subflow. The results are depicted in Fig. 31.

As described in this figure, for the short flow, i.e., the flow size is 16K, 64K and 256K, the flow completion time is in accordance with the results revealed in Section 6.2.1. Specifically, all the five algorithms shows a similar behavior when the application size is 16 KB, because this flow can complete its transmission before the second subflow is established. With the increasing flow sizes, the differences between the algorithms become larger. In the WLAN+3G scenario where the paths are asymmetric, DAPS performs worst because of the high retransmissions depicted in Fig. 32, followed by OTIAS, which cannot react to network dynamics as quickly as other three algorithms either. BLEST, minRTT and DMPTCP perform similarly in terms of completion time while DMPTCP provides an improvement of 12.1% and 2% compared to minRTT and BLEST, respectively, benefit from increased contribution of 3G link and reduced retransmissions.

In addition, we can also obtain from the figure that DPAS still cannot reduce the flow completion time for short flows in the homogeneous scenarios (3G+3G and WLAN+WLAN) as only a small amount of data is transferred over the second subflow compared to other algorithms as shown in Fig. 33. Besides, we find that BLEST performs worse compared to minRTT in the 3G+3G scenario. To investigate the reasons, the retransmission ratio as well the contribution of the second subflow

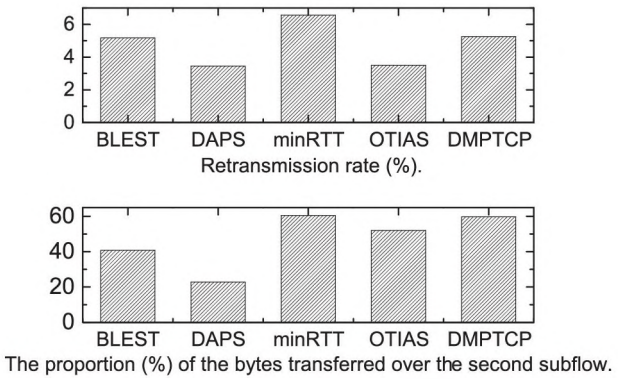


Fig. 33. The retransmission rate and the proportion of the bytes transferred over the 3G link when the flow size is 64 KB in the 3G+3G scenario.

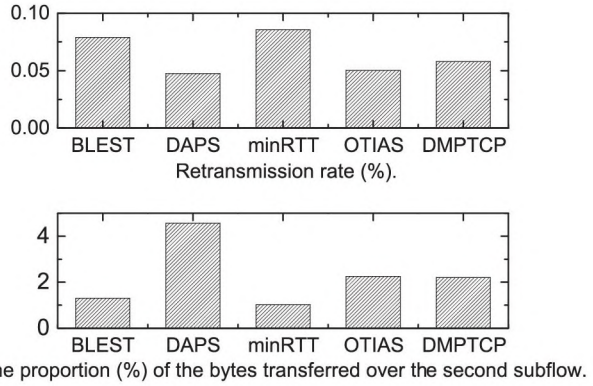


Fig. 34. The retransmission rate and the proportion of the bytes transferred over the 3G link when the flow size is 8 MB in the 3G+3G scenario.

is taken into consideration. The results are depicted in Fig. 33 which considering the flow with the size of 64 KB as an example. According to the figure, we can derive that the main reason for the poor performance of BLEST is the smaller proportion of the bytes transferred over the second subflow compared to minRTT.

For the long flows with the flow size of 8 MB, the results presented in Fig. 31 reveal that each algorithm in the WLAN+3G scenario behaves similar as that in Fig. 29. That's, the performance of DAPS and OTIAS is relatively poor with an application Goodput reduction of 20.39% and 11.72%, respectively compared to the default MPTCP minRTT scheduler in this heterogeneous scenario. Meanwhile, DMPTCP provides an application Goodput improvement up to 3.3% and 21.3% compared to BLEST and minRTT. Furthermore, the obtained Goodput of each algorithm pains a different picture in the homogeneous scenarios, e.g., WLAN+WLAN, 3G+3G, where the two paths have the same network configurations. The most obvious is that OTIAS outperforms minRTT. Take the 3G+3G scenario as an example. The retransmissions and contribution of the second subflow is illustrated in Fig. 34. According to the figure, the performance improvement of OTIAS is benefit from the reduced retransmissions and the increased proportion of the bytes transferred over the second subflow. In addition, DAPS also fails to achieve the capacity aggregation in this situation mainly because of the high retransmissions. BLEST and minRTT show similar performance. That's, a smaller amount of data is assigned to the second subflow and more retransmissions occur, causing an application Goodput reduction of about 3% compared to OTIAS and DMPTCP.

Above all, the proposed DMPTCP algorithm takes both the network configurations and the application information into consideration when scheduling packets. This guarantees DMPTCP can distribute data over each subflow adaptively to achieve performance improvements compared to the state-of-the-art scheduling algorithms in both heterogeneous and homogeneous scenarios.

7. Conclusions

MPTCP utilizes all available subflows to achieve efficient resource usage, which leads to increased delays for short flows especially when the paths have heterogeneous characteristics. In this paper, we conduct an in-depth study of MPTCP to find the root reasons and propose DMPTCP to alleviate this issue. DMPTCP first constantly monitors each path's status information, and then compares and sorts all available paths. Finally, it adaptively transmits data packets over a selected set of paths based on the MPTCP analytical model. By limiting the number of utilized paths for short flows, DMPTCP improves the short flow completion time while maintaining high performance for long-lived flows. The results obtained from extensive experiments demonstrate the effectiveness of DMPTCP. Future investigation will focus on exploring an enhanced path selection algorithm based on DMPTCP for the scenario when the application size cannot be obtained. Additionally, an improved slow start algorithm may also taken into consideration for MPTCP with short flows to alleviate the performance degradation when the subflows competing for the common bottleneck links as analyzed in the experiments.

Acknowledgement

This work is supported by the National Natural Science Foundation of China under Grants (61602171) and Scientific Research Fund of Hunan Provincial Education Department (17C0960).

References

Barik, R., Welzl, M., Ferlin, S., Alay, O., 2016. Lisa: a linked slow-start algorithm for MPTCP. In: IEEE ICC, pp. 1–7.
Barré, S., Paasch, C., Bonaventure, O., 2011. Multipath TCP: from theory to practice. In: ACM NETWORKING, pp. 444–457.
Cao, Y., Xu, M., Fu, X., 2012. Delay-based congestion control for multipath TCP. In: 20th IEEE International Conference on Network Protocols (ICNP), pp. 1–10.
Chen, Y.-C., Lim, Y.-s., Gibbens, R.J., Nahum, E.M., Khalili, R., Towsley, D., 2013. A measurement-based study of MultiPath TCP performance over wireless networks. In: ACM IMC, pp. 455–468.
China Internet Network Information Center, 2017. China statistical report on internet development. http://cnnic.cn/hlwfzyj/hlwxzbg/hlwtjbg/201701/P020170123364672657408.pdf [Accessed 3 july 2017].
Cordero, J.A., 2016. Multi-path TCP performance evaluation in dual-homed (wired/wireless) devices. J. Netw. Comput. Appl. 70, 131–139.
Dong, P., Wang, J., Huang, J., Wang, H., Min, G., 2016. Performance enhancement of Multipath TCP for wireless communications with multiple radio interfaces. IEEE Trans. Commun. 64 (8), 3456–3466.
Ferlin, S., Alay, Dreibholz, T., Hayes, D.A., Welzl, M., 2016a. Revisiting congestion control for multipath TCP with shared bottleneck detection. In: IEEE INFOCOM, pp. 1–9.
Ferlin, S., Alay, Mehani, O., Boreli, R., 2016b. Blest: blocking estimation-based MPTCP scheduler for heterogeneous networks. In: IFIP Networking Conference (IFIP Networking) and Workshops, pp. 431–439.
Ford, A., Raiciu, C., Handley, M., Bonaventure, O., 2013. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824.
Google, 2017. mptcp-ns3. http://code.google.com/p/mptcp-ns3/.
Habib, S., Qadir, J., Ali, A., Habib, D., Li, M., Sathiaseelan, A., 2016. The past, present, and future of transport-layer multipath. J. Netw. Comput. Appl. 75, 236–258.
Hassayoun, S., Iyengar, J., Ros, D., 2011. Dynamic window coupling for multipath congestion control. In: 19th IEEE International Conference on Network Protocols (ICNP), pp. 341–352.
Hwang, J., Yoo, J., Hurtig, P., Alay, O., F, S., Yedugundla, K., 2015. Reducing transport latency using multi-path protocols. In: European Conference on Communications (EUCNC) Special Session on Reducing Latency.
Hwang, J., Yoo, J., 2015. Packet scheduling for multipath TCP. In: Seventh International Conference on Ubiquitous and Future Networks, pp. 177–179.
Khalili, R., Gast, N., Popovic, M., Le Boudec, J.-Y., 2013. MPTCP is not pareto-optimal: performance issues and a possible solution. IEEE/ACM Trans. Netw. 21 (5), 1651–1665.
Kheirkhah, M., Wakeman, I., Parisis, G., 2016. Mmptcp: a multipath transport protocol for data centers. In: IEEE INFOCOM, pp. 1–9.
Kimura, B.Y.L., Lima, D.C.S.F., Loureiro, A.A.F., 2017. Alternative scheduling decisions for multipath tcp. IEEE Commun. Lett. 21 (11), 2412–2415.
Kuhn, N., Lochin, E., Mifdaoui, A., Sarwar, G., Mehani, O., Boreli, R., 2014. Daps: intelligent delay-aware packet scheduling for multipath transport. In: IEEE ICC, pp. 1222–1227.
Ma, L., Yu, F.R., Leung, V.C.M., 2007. Performance improvements of mobile SCTP in integrated heterogeneous wireless networks. IEEE Trans. Wireless Commun. 6 (10), 3567–3577.
Ni, D., Xue, K., Hong, P., Zhang, H., Lu, H., 2015. Ocps: offset compensation based packet scheduling mechanism for multipath TCP. In: IEEE ICC, pp. 6187–6192.
Oh, B.-H., Lee, J., 2015. Constraint-based proactive scheduling for MPTCP in wireless networks. Comput. Netw. 91, 548–563.
Paasch, C., Ferlin, S., Alay, O., Bonaventure, O., 2014. Experimental evaluation of Multipath TCP schedulers. In: ACM SIGCOMM, pp. 27–32.
Peng, Q., Walid, A., Hwang, J., Low, S., 2016. Multipath TCP: analysis, design, and implementation. Networking. IEEE/ACM Trans. 24 (1), 596–609.
Peng, Q., Walid, A., Low, S.H., 2013. Multipath TCP algorithms: theory and design. ACM SIGMETRICS Perform. Eval. Rev. 41 (1), 305–316.
Raiciu, C., Barre, S., Pluntke, C., Greenhalgh, A., Wischik, D., Handley, M., 2011. Improving datacenter performance and robustness with Multipath TCP. In: ACM SIGCOMM, pp. 266–277.
Raiciu, C., Paasch, C., Barre, S., Ford, A., Honda, M., Duchene, F., Bonaventure, O., Handley, M., 2012. How hard can it be? designing and implementing a deployable Multipath TCP. In: 9th USENIX Conference on Networked Systems Design and Implementation (NSDI), p. 29.
Sarwar, G., Boreli, R., Lochin, E., Mifdaoui, A., Smith, G., 2013. Mitigating receiver's buffer blocking by delay aware packet scheduling in multipath data transfer. In: 27th International Conference on Advanced Information Networking and Applications Workshops, pp. 1119–1124.
Wischik, D., Raiciu, C., Greenhalgh, A., Handley, M., 2011. Design, implementation and evaluation of congestion control for multipath TCP. In: 8th USENIX Conference on Networked Systems Design and Implementation (NSDI), pp. 99–112.
Yang, F., Wang, Q., Amer, P.D., 2014. Out-of-order transmission for in-order arrival scheduling for multipath TCP. In: 28th International Conference on Advanced Information Networking and Applications Workshops, pp. 749–752.
Yedugundla, K., Ferlin, S., Dreibholz, T., Alay, Ö., Kuhn, N., Hurtig, P., Brunstrom, A., 2016. Is multi-path transport suitable for latency sensitive traffic? Comput. Netw. 105. 1–21.

Pingping Dong received her B.S., M.S. and Ph.D degree from the School of Information Science and Engineering at Central South University, P. R. China. Currently she is a teacher in the College of Information Science and Engineering, Hunan Normal University, Changsha, P.R. China. Her research interests include protocol optimization and protocol design in wide area networks (WANs) and wireless local area networks (WLANs).

Wenjun Yang is currently a student pursuing his Master degree at College of Information Science and Engineering, Hunan Normal University, Changsha, P.R. China. His current research interests are focused on protocol optimization for heterogeneous networks.

Wensheng Tang received his B.S. degree from Hunan Normal University, Changsha, China in 1992, and received his M.S. degree and Ph.D. degree from National University of Defense Technology, Changsha, China in 1997 and in 2009, respectively. His research interests focus on the protocol optimization and cloud computing. He is currently a professor of Hunan Normal University, Changsha, China.

Jiawei Huang obtained his PhD (2008) and Masters degrees (2004) from the School of Information Science and Engineering at Central South University. He also received his Bachelors (1999) degree from the School of Computer Science at Hunan University. He is now an associate professor in the School of Information Science and Engineering at Central South University, China. His research interests include performance modeling, analysis, and optimization for wireless networks and data center networks.

Haodong Wang received the Ph.D. degree in computer science from the College of William and Mary, Williamsburg, VA, USA, in 2009. He is currently an Associate Professor with the Department of Electrical Engineering and Computer Science, Cleveland State University. His research interests include security and privacy, parallel computing, cloud computing, wireless networks, sensor networks, pervasive computing systems, and software defined radio. He is a member of the ACM.

Yi Pan received his Bachelor and Master degrees in computer engineering from Tsinghua University, China, in 1982 and 1984, respectively, and his Ph.D. degree in computer science from the University of Pittsburgh, USA, in 1991. He is currently a Professor and Chair of the Department of Computer Science and Professor of the Department of Computer Information Systems at Georgia State University. His research interests include parallel and cloud computing, wireless networks, and bioinformatics.

Jianxin Wang received the Ph.D. degree in computer science from Central South University, China, in 2001. Currently, he is a professor at School of Information Science and Engineering, Central South University, China. His current research interests include algorithm analysis and optimization, computer network and bioinformatics. He has published more than 100 papers in various International journals and refereed conferences. Dr. Wang is serving as the program committee chair or member of several international conferences. He is a senior member of Institute of Electrical and Electronics Engineers (IEEE).