IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 29, 2018, accepted December 7, 2018, date of publication January 1, 2019,
date of current version January 16, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2889339

Tuning the Aggressive Slow-Start Behavior of MPTCP for Short Flows

PINGPING DONG[1], WENJUN YANG[1], KAIPING XUE[2], (Senior Member, IEEE), WENSHENG TANG[1], KAI GAO[3], AND JIAWEI HUANG[4], (Member, IEEE)

[1]College of Information Science and Engineering, Hunan Normal University, Changsha 410081, China
[2]Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei 230027, China
[3]College of Automotive and Mechanical Engineering, Changsha University of Science and Technology, Changsha 410114, China
[4]School of Information Science and Engineering, Central South University, Changsha 410083, China

Corresponding author: Kaiping Xue (kpxue@ustc.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61602171, in part by the Youth Innovation Promotion Association CAS under Grant 2016394, and in part by the Scientific Research Fund of the Hunan Provincial Education Department under Grant 17C0960.

ABSTRACT With the widespread availability of multi-homed devices, enabling multiple paths by utilizing multipath TCP (MPTCP) in current networks is a common practice to improve the performance and robustness. Although MPTCP improves both bandwidth efficiency and network reliability, the transmission performance for short flows can become worse than regular single path TCP when the concurrent subflows transferred through a shared bottleneck due to MPTCP's aggressive slow start behavior, which is uncoupled, and each subflow behaves independently as regular TCP, affecting MPTCP and concurrent traffic at the bottleneck. In this paper, we first reveal that the MPTCP's aggressive behavior in slow start causes timeouts and throughput collapse. We further present the design and implementation of GSAM, which employs the theoretical analysis to derive the appropriate threshold for smoothing the congestion window growth in GSAM according to the network conditions in slow start phase and leverages congestion detection and control at end-host to avoid buffer overflow under concurrent MPTCP connections with multiple subflows sharing the bottleneck. The experimental results based on the real implementations show that the GSAM reduces the completion time by up to 80% while retaining high Goodput for large flows as MPTCP.

INDEX TERMS Multipath TCP, completion time, RTO, slow start, short flow.

I. INTRODUCTION

With the increased popularity of hand-held devices equipped with multiple heterogeneous radio interfaces and multi-homing capable data-centers connected to the Internet with several network access links [1], multiple paths exist between the server machines and end-hosts [2], [3]. This motivates the research and industry effort in multipath transport protocols [4]–[7].

Multipath TCP (MPTCP) is an ongoing effort by the Internet Engineering Task Force (IETF), which is a set of extensions to TCP. By allowing the use of multiple network paths for a single data stream, MPTCP increases robustness during times of path failure, and potentially achieves higher end-to-end throughput [8], [9]. As more paths are utilized compared to regular single path TCP, MPTCP cannot fairly share the bottleneck with TCP if MPTCP runs the independent congestion control on each subflow [10]. To build MPTCP protocols compatible with the regular TCP, the existing MPTCP congestion control protocols couple the congestion control algorithms running on different subflows by linking their increase functions, and dynamically controls the overall aggressiveness of the multipath flow [11], [12].

However, this only works in the congestion avoidance phase [13], [14]. In slow start phase, each subflow of MPTCP behaves as regular TCP and the congestion window increases exponentially every RTT round [15]. That is, all subflows independently double their congestion windows (cwnd) in slow-start, resulting in also doubling MPTCP's compound cwnd which is calculated as the sum of the cwnd of each subflow. The compound MPTCP cwnd can briefly increase by a large number, causing high queue occupancy and even frequent packet loss events at shared bottlenecks, resulting in suboptimal MPTCP performance.

Moreover, most of the TCP sessions in the today's Internet is constituted by short flows (e.g. web requests) and more than 40% of the web transfers are of size smaller

than 1MB [14]. For these short flows, TCP will likely never leave the slowstart (SS) phase, therefore SS behavior becomes of critical importance for the performance. Given the fact that it is usually consumes a long time for short flows to recover from the serious packet loss, the congestion collapse caused by the MPTCP's competing subflows leads to longer flow completion time of MPTCP compared to regular TCP.

To eliminate this behavior, Barik *et al.* [14] proposed a linked slow-start algorithm (LISA) where each new subflow takes a credit from an existing subflow needed for its own initial congestion window. However, the performance degradation of MPTCP is mainly caused by the aggressive growth of concurrent subflows'cwnd and LISA cannot solve this problem fundamentally as shown in Section II. To address this issue, Wang *et al.* [16] presented a Coupled Slow-Start (CSS) Algorithm. CSS couples the Slow-Start Threshold (ssthresh) of different subflows and links the exponential growth of subflows' congestion windows to ensure the fairness and reduce the packet loss. However, the congestion window in CSS is also coupled as LISA to make sure its total cwnd is the same as that of TCP. When the subflows do not share the bottleneck, its performance may also degrade due to the reduced congestion window size compared to regular MPTCP as well as TCP as validated in Section V.

In this paper, we argue that the main reason for the performance degradation of MPTCP when dealing with the short flows is the router buffer overflow caused by the aggressive slow start behavior. Thus, we propose a Gentle Slow stArt scheme for MPTCP (GSAM), which smoothes the aggressive increasing of congestion window in slow start phase of MPTCP to avoid the buffer overflow which causes packet losses and even timeout events. Meanwhile, it maintains MPTCP's high performance when the subflows are disjoint. The main contributions of this paper are as follows:

- Based on the in-depth study of MPTCP's slow start mechanism, we propose a Gentle Slow stArt scheme for MPTCP, namely GSAM for concurrent multipath transfer. To effectively use all available paths with guarantee to avoid the router buffer overflow which causes packet losses and even timeout events when transmitting short flows, GSAM finds the appropriate thresholds to smooth the aggressive slow start behavior.
- The main challenges in the proposed GSAM algorithm are the congestion window growth tuning timing and the tuning magnitude. To this end, this paper uses the theoretical analysis to derive the appropriate threshold for smoothing the congestion window growth in GSAM according to the network conditions, and leverages congestion detection and control at end-host to avoid buffer overflow under concurrent MPTCP connections with multiple subflows sharing the bottleneck.
- We validate the effectiveness of GSAM by implementing GSAM on Linux kernel and setting up a testbed consisting of two subflows. Extensive experiments reveal that GSAM can decrease the completion time of short flows by up to 35.73% compared with the MPTCP's default scheduler minRTT. While for long flows, up to 21.3% of this revenue benefits from the reduced retransmission and RTOs.

The rest of the paper is organized as follows. The design motivation of GSAM is presented in Section 2. Section 3 discusses the related works. Section 4 describes the details of GSAM. We evaluate GSAM with the Linux testbed in Section 5. Finally, Section 6 concludes the paper.

II. MOTIVATION

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

In the current existing MPTCP, the slow start of each subflow behaves the same as an independent TCP connection. That's, the congestion window is increased by one on receiving a new ACK, doubling the window size each round-trip time. This aggressiveness behavior can bring the compound congestion window of the MPTCP flow to a large value, causing TCP timeouts and throughput collapse when the subflows competing the same bottleneck.

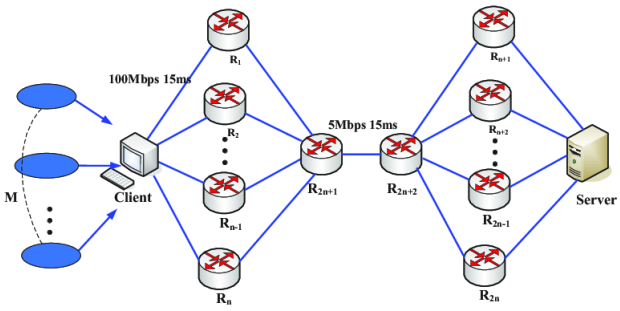


FIGURE 1. The topology to investigate the impact of the aggressive slow-start behavior of MPTCP when the subflows share a common bottleneck. The file size is 55KB, the bottleneck capacity is 5Mbps, and the router buffer size is 50 packets.

In order to illustrate the issues, we conduct our experiments with the topology shown in Fig. 1 based on NS3 testbed where two concurrent flows with the size of 55KB run between the client and server which are connected through N subflows sharing a common bottleneck with 5Mbps bandwidth and 150ms RTT.

Fig. 2 illustrates the completion time with varying number of sublfows. As follows from this figure, when the number of the competing subflows is small, i.e., 2 and 4, MPTCP as well as other two multipath protocols, namely, GSAM and LISA outperforms regular TCP benefit from the increased subflows. Besides, MPTCP also performs better compared to GSAM and LISA in this situation as it increases the congestion window faster as illustrated in Fig. 3.

Fig. 3 describes the growth of the congestion window of each algorithm over each subflow varying with time when there are two subflows. The path manager of MPTCP, whose

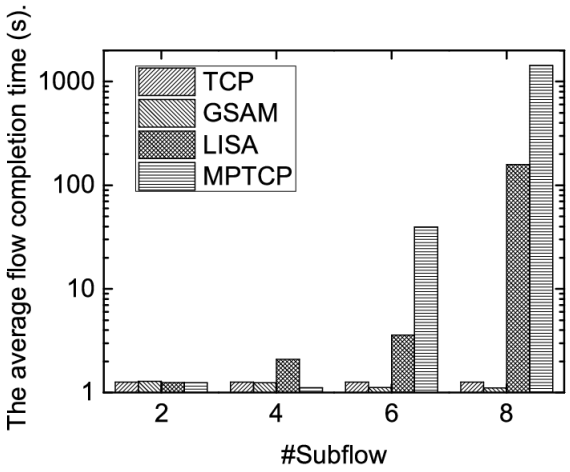


FIGURE 2. The flow completion time when two concurrent flows with the size of 55KB transfer across a common bottleneck of 5Mbps with the RTT of 150ms. The y-axis is log-scaled.

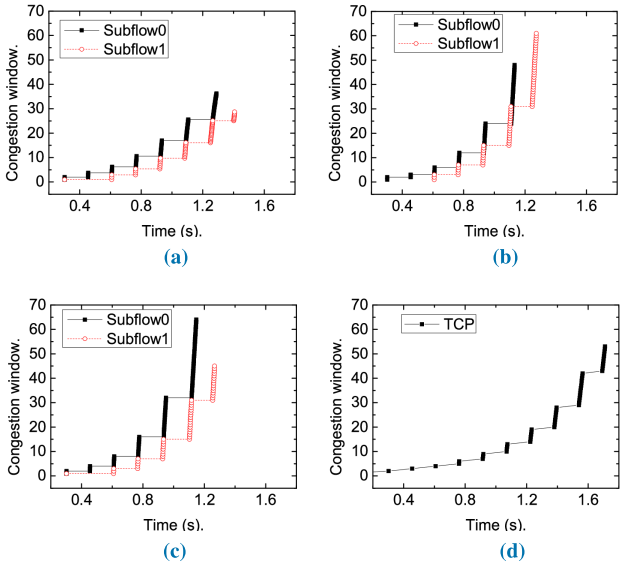


FIGURE 3. The congestion window over each subflow of each algorithm according to the results shown in Fig. 2. (a) GSAM. (b) LISA. (c) MPTCP. (d) TCP.

place is shown in Fig. 4, controls the establishment of new subflows and works as follows [14]. It starts a MPTCP connection by establishing an initial TCP subflow (Subflow0) with a standard TCP 3-way handshake. This initial subflow works the same as regular TCP in slow start, doubling its window on receiving a new ACK. Thus, it has the same congestion window as TCP. Moreover, in MPTCP, if the peer host supports Multipath TCP, it will advertise all additional IP addresses to the connection initiator during this procedure. Then, the additional subflows are able to join in the Multipath TCP connection and can start sending once they are established [14], making its compound congestion window, the sum of the congestion window of all the subflows' is larger than regular TCP.

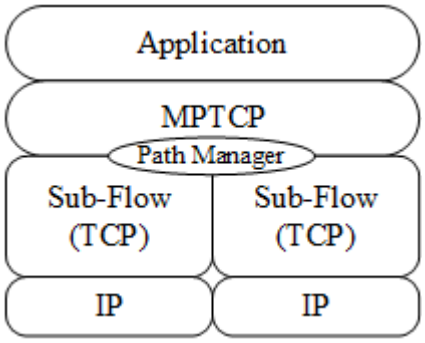


FIGURE 4. The architecture of MPTCP.

Take the MPTCP as an example. Its compound congestion window is about 8 at 0.6098s. However, the congestion window of regular TCP is only 4 at the same time. In addition, the congestion window of LISA and GSAM is about 7 at this time, which is a little slower than MPTCP. The reason lies in that LISA begins its second subflow by taking credits from the first subflow while GSAM has a more gentle window growth mode. As a result, MPTCP performs best in this situation, followed by LISA and GSAM, and TCP has the largest flow completion time.

It's to note that there is a contrast with LISA with 4 subflows, where LISA performs worse than regular TCP. To provide a more thorough analysis and understanding of this phenomenon, we draw the graphes that depict the congestion window of each algorithm varying with time when there are 4 subflows. The results are shown in Fig. 5 and Fig. 6.

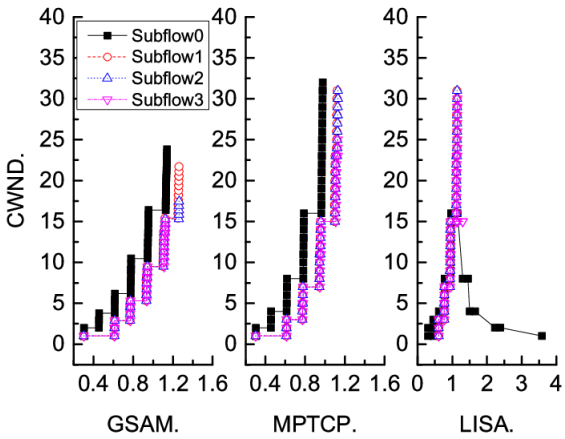


FIGURE 5. The congestion window over each subflow of each algorithm when there are 4 subflows.

According to the window evolution of each algorithm, the reason why LISA with 4 subflows breaks the trend compared to the other cases in this situation is because CWND reductions exist with LISA, which indicate packet loss events occur. We further introduce the workflow of LISA as follows. LISA first finds that subflow0 has the largest congestion window. Then, data packets are taken from it and given to subflow1, subflow2 and subflows3 as their current

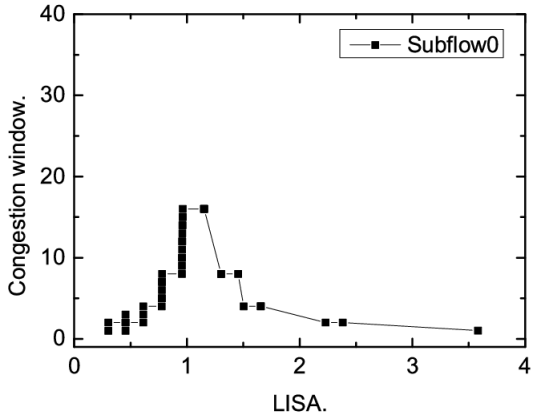


FIGURE 6. The congestion window over subflow0 of LISA when there are 4 subflows.

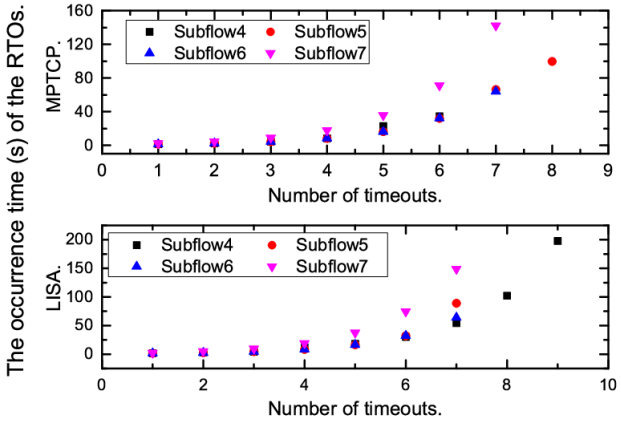


FIGURE 7. The occurrence time of retransmission timeout over each subflow of LISA and MPTCP in the network scenario shown in Fig. 2.

```
MpTcpSubFlow::MpTcpSubFlow()
    : routeId (0), state (CLOSED), phase (Slow_Start),
      sAddr (Ipv4Address::GetZero ()), sPort (0),
      dAddr (Ipv4Address::GetZero ()), dPort (0),
      olf (0), mapDSN (0), lastMeasuredRtt (Seconds (0.0))
{
    connected   = false;
    TxSeqNumber = rand() % 1000;
    RxSeqNumber = 0;
    bandwidth   = 0;
    cwnd        = 1;                    // congestion window is initialized to one segment
```

FIGURE 8. In NS3, the congestion window is initiated to one segment.

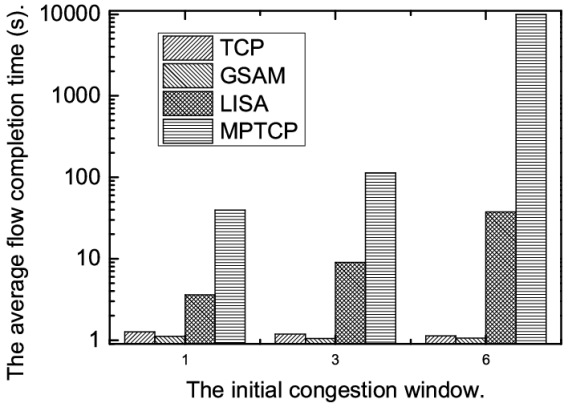


FIGURE 9. The flow completion time of each algorithm when two concurrent flows with the size of 55KB transfer across a common bottleneck of 5Mbps with the RTT of 150ms. The number of subflows is 6. The y-axis is log-scaled.

congestion window. This makes Subflow0 increase CWND much slower compared to other algorithms at the same time as shown in Fig. 5 and Fig. 6 (the solid line). On the contrary, in MPTCP, the congestion window of Subflow0 grows fast and attains 31 at 0.97696s. Then, no more packets are assigned to subflow0. Although the CWND of other three subflows grows exponentially to about 30 packets at 1.12s, the total data amount that stays in the network is about 87 packets.

However, in LISA, when packet loss occurs, all the four subflows are utilized to transmit packets and their CWND are 16, 31, 31 and 30, respectively. The sum of the congestion window of the four subflows is up to 108 packets. In this experiment, the buffer size is 50 packets and the BDP is about 52 packets. Thus, the maximum amount of data that allowed to stay in the network is about 102 packets.

In addition, from the Fig. 2, we can also find that the completion time of MPTCP and LISA increases sharply and is significantly higher than that of TCP and GSAM when further increasing the subflows, i.e., 6 or 8 subflows. This is due to the frequent RTOs caused by the competing subflow in the bottleneck as shown in Fig. 7.

Moreover, it is unexpected that LISA suffers and shows a similar behavior as MPTCP when the subflows is more than 2, i.e., 6 or 8. To investigate the reasons, we find that the workflow of LISA is as follows in this shared bottleneck scenario where the RTT of each subflow is same. The initial subflow (Subflow0) is first established. Then, the other subflows, namely subflow1, subflow2, subflow3, subflow4 and subflow5 are established simultaneously after the 3-way handshake of subflow0 when the number of the subflow is 6. During analyzing the trace file, we find that the congestion window of subflow0 is 3 at this time which starts from 1 in NS3 as shown in Fig. 8. Based on the workflow of LISA, subflow1 and subflow2 each takes 1 packet from subflow0 as its congestion window and the congestion window of subflow0 becomes 1. Then, no more packets can be taken from subflow0 to subflow3, subflow4 and subflow5 and these subflows behave independently as that in MPTCP, causing RTOs as described in Fig. 7.

In addition, we conduct further experiments with varying values of the initial congestion window. The results are shown in Fig. 9.

According to these results, the flow completion time of TCP and GSAM decreases with the increasing IW, which benefits from the increased network utilization. However, the increased IW leads to further performance degradation in MPTCP as well as LISA. The reason lies in that the increased congestion window exacerbates the problem of the heavy packet loss caused by the unlimited exponential growth of concurrent subflows' cwnd as analyzed above.

Based on the above analysis, we conclude that the aggressive behavior of the MPTCP's slow start is one of the main

factors that restrict the MPTCP performance especially when multiple subflows compete for the same bottleneck. These observation motivates us to design a novel slow start approach to avoid the RTOs at the bottleneck to improve the performance of MPTCP. In the rest of this paper, we present our GSAM as well as its performance validation with extensive experiments.

III. RELATED WORK AND EXISTING PROBLEMS

MPTCP can offer high bandwidth and reliability by leveraging multiple paths available between end-points. It has drawn considerable research attention. Path scheduling and congestion control are two of the focuses.

Packet scheduling policies are designed for distributing data packets over multiple paths to alleviate the packet reordering issue, and thus improve the flow completion time and network throughput. Several packet scheduling schemes have been proposed to achieve these goals, such as Blocking Estimation-based MPTCP Scheduler (BLEST) [17], Out-of-order transmission for in-order arrival scheduling policy (OTIAS) [18], Delay-Aware Packet Scheduling (DAPS) [19], Forward Prediction Scheduling (FPS) [20], Fine-grained Forward Prediction based Dynamic Packet Scheduling ($F^2P - DPS$) [21], Offset Compensation based Packet Scheduling (OCPS) [22], DMPTCP [23], Receive Buffer Pre-division based flow control mechanism (RBP) [24], Forward Prediction based Dynamic Packet Scheduling and Adjusting with Feedback (DPSAF) [25].

The congestion control algorithm of MPTCP aims to improve throughput, be friendly to traditional TCP and balance congestion [26]. Balance Congestion means MPTCP should utilize the least congested path [27]. Several congestion control schemes have been proposed to achieve these principles. The coupled congestion-control scheme defined in [27] is an adaptation of the NewReno algorithm for multipath transfer which increases or decreases the congestion window by considering the status of all subflows. However, it is non-pareto optimality. Thus, the MPTCP-OLIA algorithm [28] was proposed to solve the problem by improving the mechanism of the congestion window increasement. Meanwhile, BALIA [29] was raised to provide a better balance between TCP friendliness, responsiveness, and window oscillation. Furthermore, our proposal, mVeno [30], improves MPTCP performance in lossy wireless networks. Besides, the delay-based MPTCP scheme named wVegas [31] is proposed to achieve fine-grained load balancing.

However, these existing congestion control algorithms are mainly focus on the congestion avoidance phase which cannot solve the congestion collapse problem in the slow start phase when the concurrent MPTCP connections are high at the shared bottlenecks.

Barik *et al.* [14] propose LISA to solve the problem by coupling the MPTCP subflows during the slow-start phase. In LISA, each new subflow takes a credit from an existing subflow for its own initial congestion window to make sure that the sum of the congestion window of all subflows is not larger than that of TCP flow. However, when there are multiple subflows, LISA may also show poor performance like regular MPTCP as the subflows can be opened at the same time, and thus cannot take cwnd from the existing subflow as its congestion window can be quite small (i.e, ≤ 6 as defined in the algorithm). Besides, the window growth of LISA can be slower than regular MPTCP, causing suboptimal performance when the subflows are disjoint.

In the study [16], the authors present a Coupled Slow-Start algorithm named CSS. CSS resets the Slow-Start Threshold (ssthresh) of different subflows, aiming to only give the MPTCP connection as much throughtput as that in TCP when exiting Slow-Start. Besides, to ensure the fairness and reduce the burstiness of Slow-Start, CSS links the exponential growth of subflows' congestion windows by slowing down the growth of exiting subflows' cwnd when a new subflow joins. However, when the subflows do not share the bottleneck, its performance may also degrade due to the reduced congestion window size compared to regular MPTCP as validated in the paper.

In this paper, we argue that the main reason for the congestion collapse is that when multiple subflows share the same bottleneck, the exponential growth of the congestion window of each subflow often misleads the MPTCP sender to send too many packets too quickly, thus causing a severe router buffer overflow at the bottleneck link. As a result, retransmission timeout and throughput collapse happen. This motivated us to investigate a novel approach smoothing the aggressive increasing of congestion window in slow start phase of MPTCP. In the rest of this paper, we present our GSAM and evaluate its performance in various conditions.

IV. GSAM

In this section, we firstly describe the design details of GSAM. Then, the congestion window tuning mechanism is present. Finally, based on the theoretical analysis of the slow start behavior, we give a guideline for determining the threshold that is used for smoothing the exponential expansion of the congestion window.

A. DESIGN DETAILS

The design goal of GSAM is to tune the aggressive slow start behavior of MPTCP to avoid the throughput collapse when all the subflows sharing the same bottleneck. To this end, however, GSAM faces two key challenges that (i) GSAM should obtain the accurate congestion level of each subflow in slow start stage and (ii) GSAM should smooth the exponential expansion of the congestion window, while ensuring high utilization of bottleneck link.

To achieve these goals, GSAM splits the regular slow start mechanism into two phases based on the RTT threshold value K. Firstly, GSAM behaves the same as regular MPTCP over each subflow. When the RTT value rises above the predefined threshold K, it is convinced that continuing the exponential growth can result in buffer overflow and packet

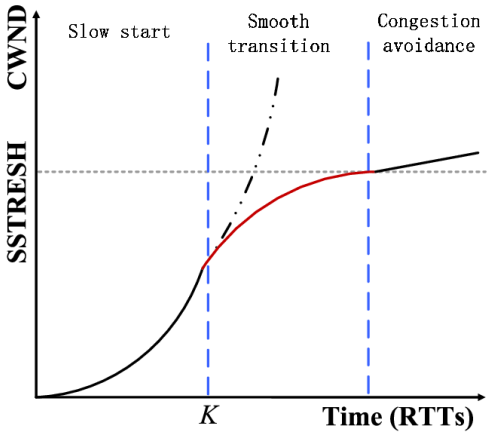


FIGURE 10. The congestion window transition over each subflow of GSAM.

losses. Then, GSAM enters the smooth transition stage as shown in Fig. 10.

B. TUNING CONGESTION WINDOW

GSAM uses RTT to detect the congestion level and tuning the congestion window accordingly over each subflow. Specifically, on receiving new ACKs, GSAM measures the current RTT, and updates three variables by the following operations: (i) updating *min_RTT*, which is the link latency without queuing delay, (ii) determining the RTT threshold K based on *min_RTT*, and (iii) calculating *smooth_RTT*, which is a smooth value of the current RTT and is calculated according to Eq. 1. In this equation, α is a smoothing parameter and is set according to the RFC 6298. These variables are kept by the TCP connection hosted by the sender.

$$Smooth\_RTT \leftarrow (1-\alpha)\cdot Smooth\_RTT + \alpha \cdot Smooth\_RTT. \tag{1}$$

Once the measured RTT exceeds K, GSAM tuning the congestion window from the exponential growth to smooth mode. As the congestion level can be represented as

$$\partial = \frac{RTT - K}{RTT}. \tag{2}$$

where bigger ∂ depicts more severe congestion, and vice versa.

Hence, when the sender of each subflow finds its RTT is larger than a predefined threshold K, its congestion window is adjusted to

$$CWND_{i+1} = CWND_i + CWND_i^{-\partial}. \tag{3}$$

According to Eq. 3, when the measured RTT is more close to the threshold K, it is more likely that the network congestion state is not severe and the congestion window can grow exponentially. Conversely, when the measured RTT is far larger than the threshold value, which implies that network congestion occurs, the congestion window should increase linearly to gradually approach the available network resource. The algorithm is illustrated in **Algorithm 1**.

Algorithm 1: The Workflow of the Proposed GSAM Algorithm on Receiving New ACKs

```
Smooth_RTT ← (1−α)·Smooth_RTT + α·Smooth_RTT;
if RTT < min_RTT then
    min_RTT ← RTT;
    Update K according to Eq. 15;
if Smooth_RTT ≥ K then
    ∂ = (RTT − K)/RTT;
    CWND_{i+1} = CWND_i + CWND_i^{−∂};
```

C. GUIDELINE FOR CHOOSING K

GSAM uses threshold K to tune the congestion window and then controls the aggressiveness of the standard slow start to avoid the buffer overflow and congestion collapse. It is a challenge to achieve both the high unitization of bottleneck link and controllable congestion window. In this subsection, we introduce how to determine the threshold K by analyzing the slow start behavior of GSAM.

Suppose that there are N concurrent MPTCP connections and each MPTCP connection consists of M subflows. The round trip time without queuing delay of a subflow is D, and K is the RTT threshold for tuning the congestion window growth mode, thus $K - D$ represents the allowed queuing latency, then we get the desired router queue length Q by

$$Q = C(K - D). \tag{4}$$

where C is the capacity of the bottleneck link.

In the meantime, the number of packets that can be allowed to stay in the network is CK, and for each subflow the allowed maximum value of window size is $(CK)/(MN)$.

Assume that at time t, the bottleneck router queue length is just equal to Q, and at the same time each subflow is in the i^{th} round's transfer, then we get the window size of each subflow in the i^{th} round by

$$W = \frac{CK}{MN}. \tag{5}$$

Since the queuing delay reaches K, all subflows start to slow down the congestion window increase rate in the next round. To avoid buffer overflow and packet losses, GSAM should make sure the buffer occupied during the last exponential growth smaller than the buffer size. That is

$$Q \leq B. \tag{6}$$

By substituting Eq. 4 to Eq. 6, we can deduce that

$$K \leq \frac{B + CD}{C}. \tag{7}$$

As K denotes the RTT threshold for tuning the congestion window growth mode, K should be no less than D which represents the round trip time without queuing. That's

$$K \geq D. \tag{8}$$

Based on Eq. 7 and Eq. 8, we can obtain Eq. 9.

$$D \le K \le \frac{B + CD}{C}. \tag{9}$$

Next, we will verify whether the deduced K can guarantee the 100% utilization of bottleneck link in this condition. According to the sliding window protocol of TCP, the network utilization can be represented as shown in Eq. 10 which is the fraction of time that the sender is busy sending

$$U = \frac{W/C}{K + L/C} \cdot M \cdot N. \tag{10}$$

where L denotes the size of a packet.

According to Eq. 5 and Eq. 9, we can derive that the congestion window size after the last exponential growth is:

$$\frac{CD}{MN} \le W \le \frac{B + CD}{MN}. \tag{11}$$

By substituting Eq. 9 and Eq. 11 to Eq. 10, we can obtain that

$$\frac{CD}{CD + L} \le U \le \frac{B + CD}{B + CD + L}. \tag{12}$$

As $L \ll CD$ and $L \ll B + CD$, we can conclude that $U \approx 1$.

However, to avoid the RTO caused by the buffer overflow, GSAM should take an action and enter the smooth transition phase during the $(i-1)^{th}$ round. In the slow start, the congestion window during the $(i-1)^{th}$ round is half of that in the i^{th}. That's,

$$W_{i-1} = \frac{W}{2}. \tag{13}$$

By substituting Eq. 5 to Eq. 11, we can get

$$\frac{CD}{2 \cdot MN} \le W_{i-1} \le \frac{B + CD}{2 \cdot MN}. \tag{14}$$

According to Eq. 5, Eq. 8 and Eq. 14, we can derive that

$$D \le \hat{K} \le \frac{B + CD}{2 \cdot C}. \tag{15}$$

Based on above analysis, we can find that K is decided by the router buffer size B, the bottleneck bandwidth C and the minimum RTT D, which are static. The number of nodes and the subflow (M and N) that shares the bottleneck do not have to be obtained in advance.

V. TESTBED EXPERIMENT

In this section, we validate the proposed GSAM algorithm by conducting experiments on our testbed in both shared and non-shared bottleneck scenario and comparing its performance with LISA [14], the default MPTCP and regular TCP. We use the publicly available Linux code of MPTCP, and also modified the Linux kernel to implement GSAM and LISA. The source code is freely available from https://github.com/zhua451/MPTCPcode/tree/master/Linux. The performance of each algorithm with different flow size, loss rate, router buffer size, as well as concurrent flows is taken into consideration.

In addition, multiple active queue management (AQM) algorithms are introduced recently. AQM differs from the FIFO when dropping packets, which can have an impact on the performance of GSAM. Thus, we also investigate how each algorithm works with the AQM algorithm.

A. TESTBED CONSTRUCTION AND EXPERIMENTAL METHODOLOGY

The deployed experiment testbed consists of two file servers, two computers with WANem and two clients, which constitutes the network topology shown in Fig. 11 by means of routing configurations.

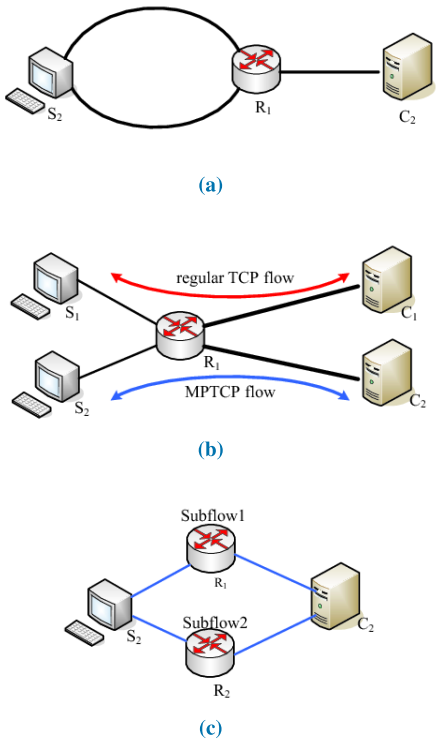


FIGURE 11. TestBed topology. (a) Shared-Bottleneck. (b) Competing-Bottleneck. (c) Non-Shared-Bottleneck.

As analyzed in section II, MPTCP's slow start is inefficient for short flows as the slow start is uncoupled and behaves the same as regular TCP, affecting MPTCP and concurrent traffic at the bottleneck. In this subsection, we first examine whether GSAM can alleviate these performance impairments of MPTCP in the shared bottleneck scenario with the topology shown in Fig. 11a and Fig. 11b. Then, we conduct experiments in the non-shared bottleneck scenario as depicted in Fig. 11c in order to see whether the reduced aggression of GSAM has the negative impact on the performance or not.

In the topology shown in Fig. 11, 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 (2.4GHz/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, 512MB RAM and 160 GB hard disk. As shown in Fig. 11, one of the servers labeled S_2 is equipped with two Gigabit network interface cards to establish two subflows between the MPTCP client C_2. We consider this as the common scenario (e.g., a client having two access networks like WiFi/4G) [14], [17]. 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 when the packet loss rate is low and fifty rounds when the packet loss probability is high. All the experimental data is captured at the clients using tcpdump and then is analyzed with wireshark.

In our Linux testbed, the bottleneck capacity is set to 5 Mbps, RTT is set to 40 ms as that in [14], which is based on the Akamai's Q1 2015 report indicating that the global average connection speed is 5 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.

B. EXPERIMENTAL RESULTS

We investigate the performance of each algorithm in three cases: a) the client's access link (downlink) is the bottleneck and it is therefore shared among the two flows (Fig. 11a), b) the regular TCP traffic competing the same bottleneck with MPTCP traffic as shown in Fig. 11b, and c) there are no shared bottlenecks between the MPTCP's two subflows (Fig. 11c). In each network scenario, the performance of each algorithm with varying router buffer size, different flow size as well as different number of concurrent flows are taken into consideration.

More specifically, the binary documents range from 50KB to 1MB. This range has been selected based on the HTTP transfer size statistics indicating that more than 40% of the HTTP transfers are of size up to 1000 KByte [14]. The router buffer size also varies from 5 packets to 100 packets and the number of concurrent MPTCP flows ranges from 1 to 15.

Note that we expect to see the benefits of GSAM in case a) and b), whereas GSAM may potentially be harmful in case c).

1) SHARED-BOTTLENECK

Fig. 12 describes the average flow completion time of each algorithm with the topology shown in Fig. 11a when the bottleneck bandwidth is 5Mbps, the RTT is 40ms and the loss rate is 0.01%.

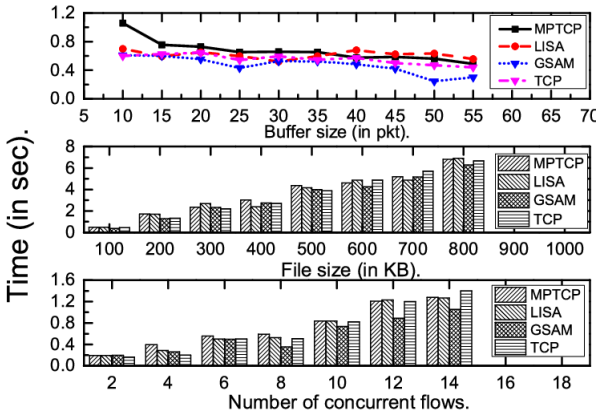


FIGURE 12. The performance of each algorithm under varying router buffer size, flow size as well as the number of concurrent flows when the bottleneck bandwidth is 5Mbps, the RTT is 40ms and the loss rate is 0.01%. The topology utilized is shown in Fig. 11a. The NCF denotes the number of concurrent flows.

According to Fig. 12, the flow completion time decreases with the increasing router buffer size where GSAM performs best, followed by TCP. In addition, LISA outperforms MPTCP when the router buffer size is small, i.e., ≤ 35. However, MPTCP performs slightly better than LISA when the router buffer size is larger than 35 pkt.

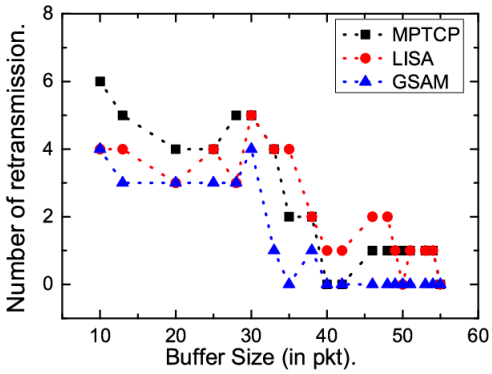


FIGURE 13. The number of retransmission of each algorithm with varying router buffer size under the network scenario shown in Fig. 12.

To reveal the reasons, we conduct further experiments. The retransmissions of each algorithm is depicted in Fig. 13. Based on this figure, we can find that the gains of GSAM is benefit from the reduced retransmission. Specifically, when the router buffer size is small, which is the constraint on the performance of each algorithm, MPTCP has the largest number of retransmission, leading to the highest flow completion time. With the increasing router buffer size, when the router buffer size is not the constraint, the number of retransmission with each algorithm is gradually reduced to 0. To further find the reason for the poor performance of LISA when the router buffer size is large, we investigate the cwnd evolution of each algorithm which is shown in Fig. 14, which reveals that the reason is due to the slower cwnd increasement compared to other algorithms.

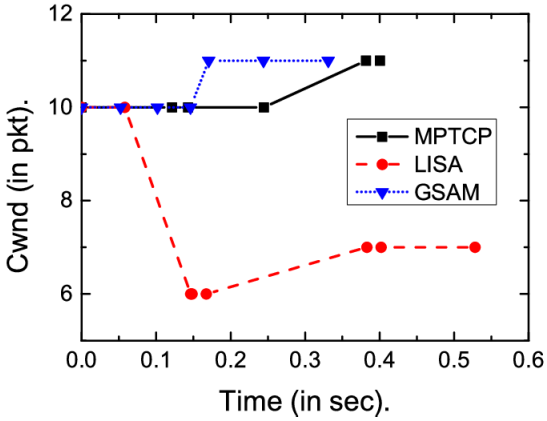


FIGURE 14. The window evaluation of each algorithm with varying router buffer size under the network scenario shown in Fig. 12.

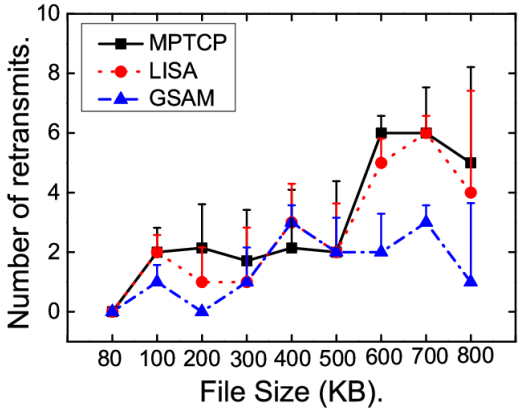


FIGURE 15. The total number of retransmissions when the flow is the last time exiting slow-start with varying flow size when the router buffer size is 20 packet and their are 5 concurrent flows under the network scenario shown in Fig. 12.

Furthermore, we can also obtain from Fig. 12 that the flow completion time becomes larger with the increasing flow size as well as the increasing number of concurrent flows. Meanwhile, each algorithm in these two conditions shows a similar behavior in the different router buffer size scenario. Fig. 15 shows the total number of retransmission when the flow is the last time exiting slow-start, which again validate that the performance gain of GSAM is benefit from the reduced retransmissions.

In addition, we also conducted experiments with different loss rate in this shared bottleneck scenario. The results are shown in Fig. 16. According to this figure, the completion time of each algorithm increases with the increasing loss rate and GSAM outperforms existing algorithms, followed by LISA and MPTCP. These results are as expected. The reason lies in that the packets are randomly dropped with the packet loss probability and these random events will influence each measurement, shifting the results both higher or lower. However, they can be eliminated through repetition and averaging [1]. In this experiment, each data point was obtained by computing the average value of the experimental results from fifty rounds of executions.

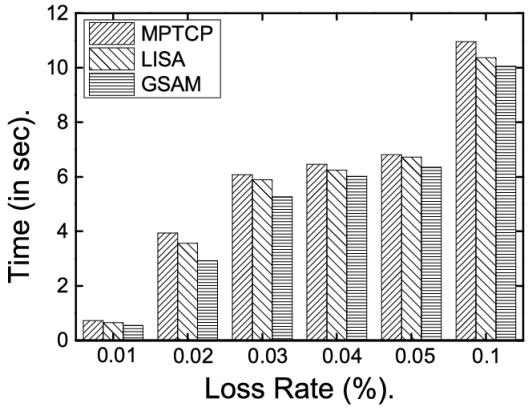


FIGURE 16. The flow completion time when 20 concurrent flows with the size of 300KB transfer across a common bottleneck of 5Mbps with the RTT of 40ms. The shared router buffer size is 20 packets.

2) COMPETING TRAFFIC

Fig. 17 depicts the performance of each algorithm when competing with regular TCP with the topology shown in Fig. 11b.

For the average flow completion time of each algorithm with varying flow size, GSAM performs best, followed by LISA when the flow size is smaller than 200KB benefit from the reduced retransmission as analyzed above. However, the LISA's completion time is slightly larger than that of MPTCP when the flow size is larger than 200KB. To reveal the reason, the cwnd evolution of each algorithm is conducted. Take the flow size of 800KB as an example, the cwnd varying with time is shown in Fig. 18. As expected, the cwnd of MPTCP increases faster than other two algorithms. GSAM increases slightly slower than MPTCP. But the sum of the congestion window of its two subflows is comparable with that of MPTCP. However, the congestion window of LISA is much smaller than GSAM as well as MPTCP especially at the start time. As depicted in Fig. 18b, when the second subflow is established at 1.2s, the congestion window of the first subflow is 4. According the the mechanism of LISA, the second subflow begins the transmission with the cwnd of 3 rather than the initial value of 10, causing small congestion window compared to other algorithms, and thus increased flow completion time.

In addition, each algorithm when competing with regular TCP traffic under varying router buffer size shows a similar behavior as that in the scenario shown in Fig. 12. Specially, when router buffer size is small ($\leq$ 20) and the buffer is the constraint, GSAM performs best, followed by LISA benefit from the reduced retransmission as shown in Fig. 19, where there are 4 times cwnd reduction with MPTCP and the number is 2 with LISA as well as GSAM. However, with the increasing router buffer size, LISA performs slightly worse than MPTCP due to the slow increase of cwnd as analyzed above.

Further, for the completion time of each algorithm increases with increasing concurrent flows, both GSAM and LISA outperforms MPTCP because of the slower

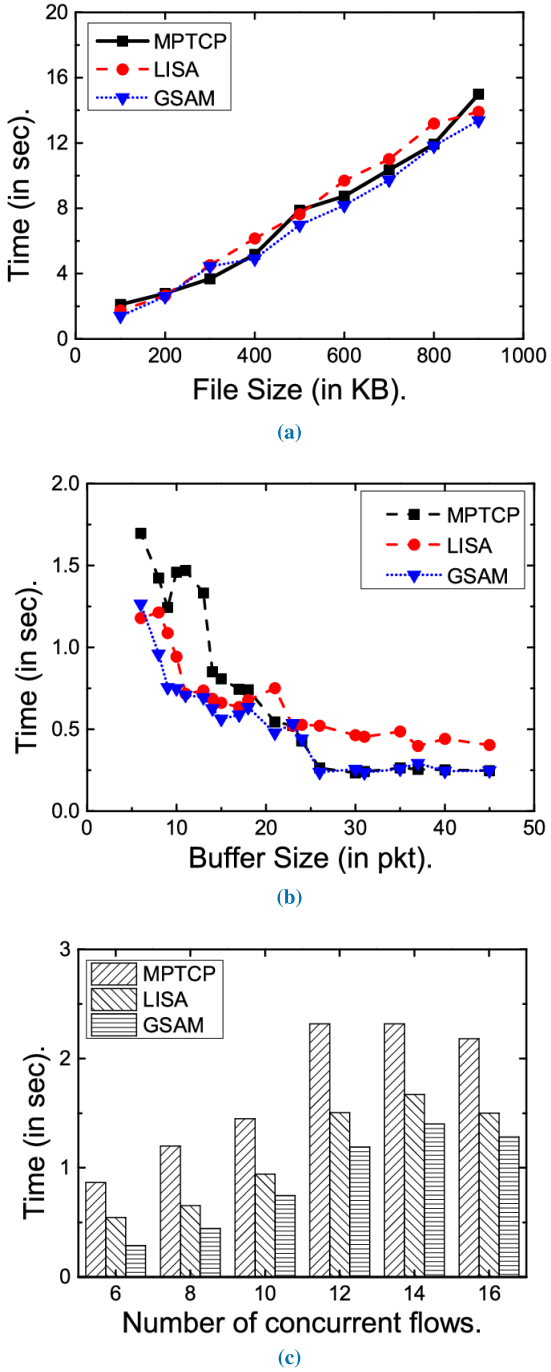


FIGURE 17. The performance of each algorithm under varying router buffer size, flow size as well as the number of concurrent flows when multipath TCP competing with regular TCP. The bottleneck bandwidth is 5Mbps, the RTT is 40ms and the loss rate is 0.01%. The topology utilized is shown in Fig. 11a. (a) File size. (b) Router buffer size. (c) Number of concurrent flows.

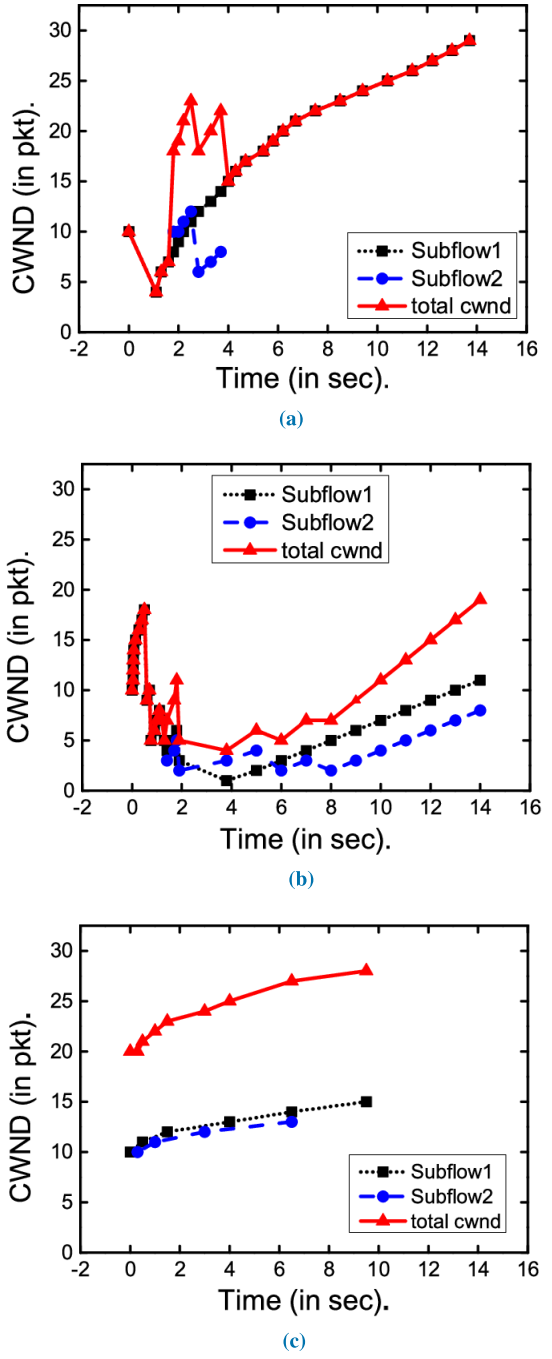


FIGURE 18. The cwnd evolution of each algorithm when competing with regular TCP traffic under the network scenario shown in Fig. 17 when the flow size is 800KB, the router buffer size is 20 and the number of concurrent flows is 5. (a) MPTCP. (b) LISA. (c) GSAM.

window increasement and reduced retransmission compared to MPTCP as shown in Fig. 20.

Finally, we investigate the impact of each algorithm on the competing regular TCP traffic. The average TCP flow completion time when competing with each algorithm is shown in Fig. 21. As depicted in this figure, TCP obtains smaller flow completion time when competing with GSAM and LISA as they slow down the window increasement, and thus alleviate the congestion collapse. This figure validate that the performance enhancement of GSAM is not at the cost of regular TCP traffic. On the contrary, the smooth transition of GSAM alleviates wasting of the network resource caused by retransmission and gives TCP a better network condition.

In this subsection, we do further experiments to investigate the performance of each algorithm when competing with

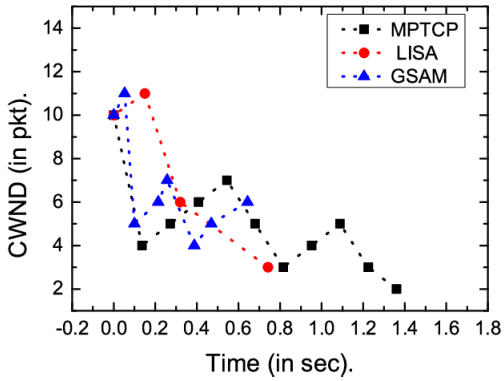


FIGURE 19. The cwnd evolution of each algorithm when competing with regular TCP traffic under the network scenario shown in Fig. 17 when the flow size is 80KB, the router buffer size is 10 and the number of concurrent flows is 5.

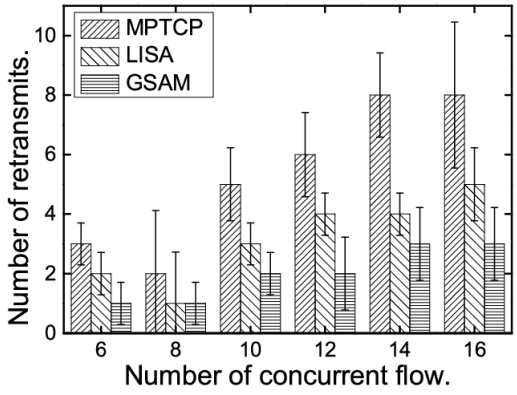


FIGURE 20. The retransmission of each algorithm when competing with regular TCP traffic with varying number of concurrent flows under the network scenario shown in Fig. 17 when the flow size is 80KB and the router buffer size is 20.

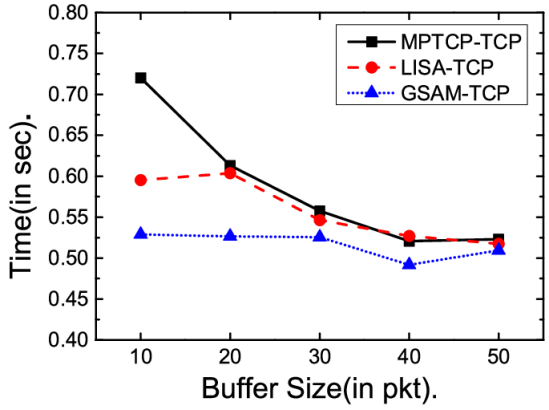


FIGURE 21. The performance of TCP when competing with each multipath algorithm under varying router buffer size with the network scenario shown in Fig. 17 when the flow size is 80KB and the number of concurrent flows is 5.

UDP flows with the network scenario shown in Fig. 11b. The experiments are done with varying shared router buffer size when there are 5 MPTCP flows competing with 5 UDP flows. The UDP traffic is generated with Iperf as shown in Fig. 24.

The results are depicted in Fig. 25. According to this figure, the flow completion time decreases with the increasing buffer size. When buffer size is small ($\leq$ 35) and the buffer is the constraint, GSAM performs best, followed by LISA benefit from slower window increasement, and thus reduced number of retransmissions. With the increasing buffer size, the difference between the flow completion time of each algorithm is getting smaller. LISA performs slightly worse than MPTCP due to the slow increase of cwnd as analyzed above.

3) NONE-SHARED BOTTLENECK

These experiments are conducted to validate whether GSAM has negative effect when there are no shared bottlenecks. The router buffer size is varying from 25 packets to 800 packets in the following experiments as the network bandwidth delay product is about 360 packets where the bandwidth is 100Mbps and the RTT of each path is 40ms. The results are shown in Fig. 22.

According to Fig. 22, the flow completion time of each algorithm decreases with the increasing router buffer size and increases with the flow size as well as the number of concurrent flows. TCP performs worst as it only utilizes one path while the other three algorithms spread packets over the two subflows simultaneously. MPTCP outperforms other algorithms as it behaves the same as regular TCP over each path in this environment. LISA performs worse compared to GSAM because of the slower window increasement as analyzed above. Similarly, GSAM shows a little performance degradation compared to MPTCP. As shown in Fig. 23, which depicts the window evolution of GSAM and MPTCP when the router buffer size is 200 packets and there are 30 concurrent flows with size of 100KB, the congestion window of MPTCP grows from 10 to 11 within 0.08s while GSAM utilizes about 0.16s.

4) IMPACT OF AQM QUEUE

The above experiments are done with the default FIFO queue mechanism. In this subsection, we do further experiments to reveal how the algorithms work with Codel, a popular AQM queue management algorithm [32]. The experiments are done with the share-bottleneck scenario as shown in Fig. 11a where MPTCP flows competing for the same bottleneck and showing performance degradation.

In Codel, there are three important parameters, namely, *limit*, *target* and *interval*. *Limit* is the hard limit on the real queue size. When this limit is reached, incoming packets are dropped. *Target* is the acceptable minimum standing/persistent queue delay. Default and recommended value is 5ms. *Interval* is used to ensure that the measured minimum delay does not become too stale. Default value is 100ms.

The proposed GSAM aims to eliminate the performance degradation of MPTCP caused by the router buffer overflow due to the unlimited exponential growth of concurrent subflows' cwnd. Thus, we first set *limit* to 20 packets to

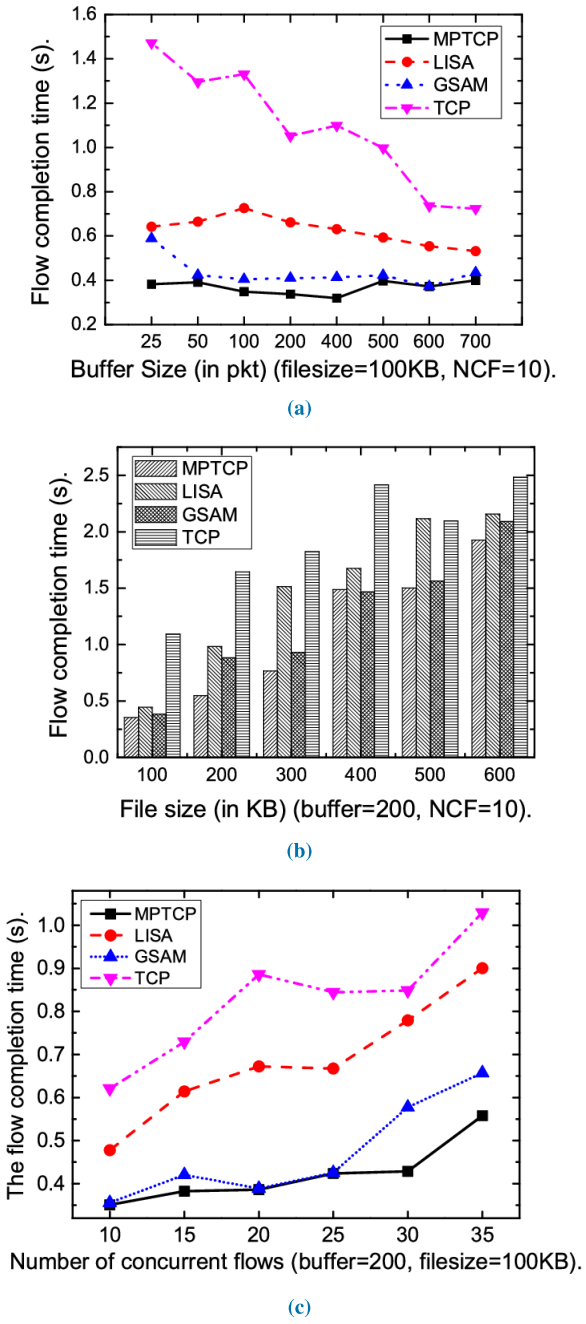


FIGURE 22. The performance of each algorithm under varying router buffer size, flow size as well as the number of concurrent flows when there are non-shared bottlenecks. The bandwidth is 100Mbps, the RTT is 40ms and the loss rate is 0.01%. The topology utilized is shown in Fig. 11c. (a) Router buffer size. (b) File size. (c) Concurrent flows.

investigate whether the proposed GSAM can achieve its goal under this buffer-restricted condition with Codel.

The results are shown in Fig. 27. According to this figure, the flow completion time decreases with the increasing *Interval* where GSAM performs best, followed by LISA.

To reveal the reasons, we conduct further experiments. The RTT evolution in time of each algorithm is depicted in Fig. 28. Based on Fig. 28, we can find that the RTT of

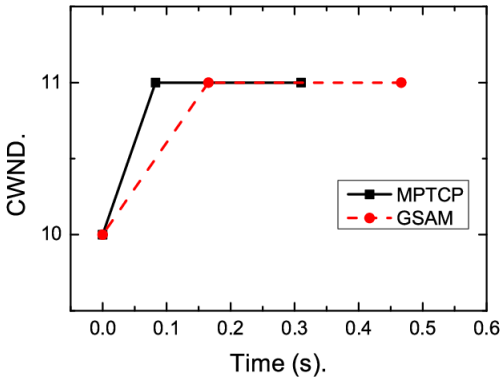


FIGURE 23. The window evolution of GSAM and MPTCP when the router buffer size is 200 packets, the number of concurrent flows is 30 and the flow size is 100KB with the network scenario shown in Fig. 22.

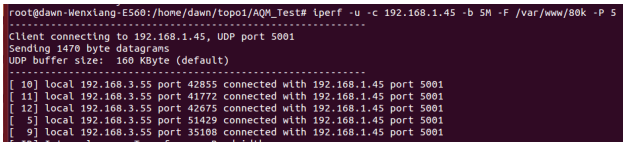


FIGURE 24. The UDP traffic is generated with Iperf by starting a server session and then the client session using 5 parallel streams.

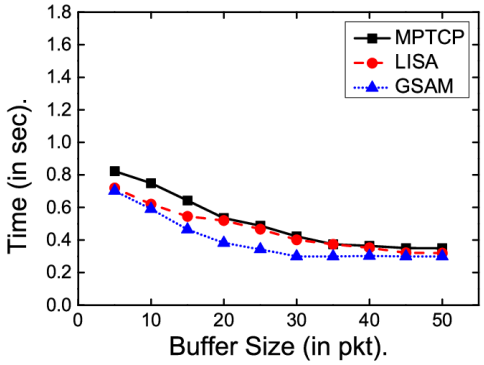


FIGURE 25. The flow completion time when the regular UDP traffic competing the shared bottleneck with MPTCP traffic.

MPTCP increases faster compared to other algorithms due to its exponentially growth of cwnd. In Codel, a packet should be dropped or not based on the comparison of RTT and *Interval*. Higher RTT means higher packet drop probability. Thus, MPTCP shows poor performance in this scenario.

Both LISA and GSAM can alleviate this phenomenon and GSAM performs better. The reason is as follows. LISA proposes to subtract one existing subflow' cwnd by the Initial Window (IW) of the newly added subflow. However, the packet loss is mainly caused by the unlimited exponential growth of concurrent subflows' cwnd and LISA fails to solve it actually. Meanwhile, GSAM starts to slow down the growth before packet loss occurs and shows the smallest RTT.

Then, we further conduct experiments by varying the value of *limit* when the *Interval* is 100ms as recommend. The results are shown in Fig. 29. According to this figure,

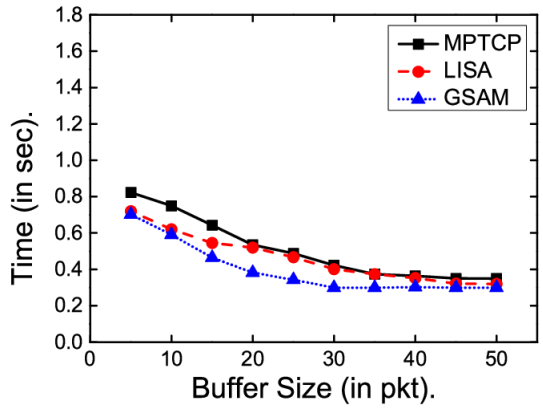


FIGURE 26. The flow completion time when the regular UDP traffic competing the shared bottleneck with MPTCP traffic.

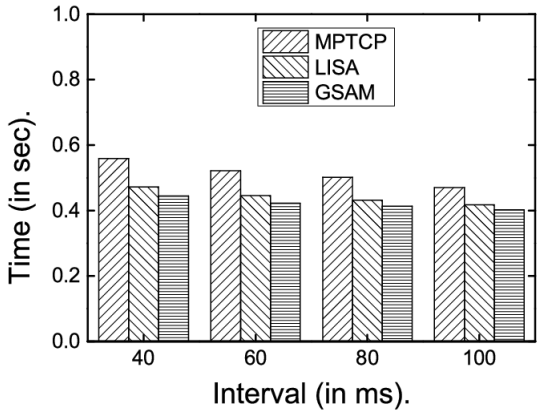


FIGURE 27. The flow completion of each algorithm with Codel under different *interval* values when the *limit* is 20, the flow size is 300KB and the number of concurrent flows is 20.

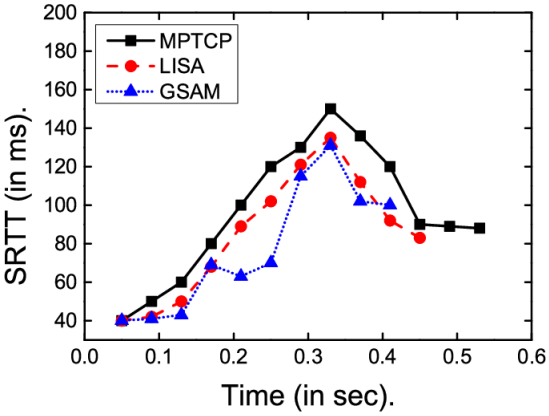


FIGURE 28. The RTT evolution in time of each algorithm corresponding to the results shown in Fig. 27 when the value of *Interval* is 40ms.

the algorithms show similar behavior as they work with the FIFO queues. Specifically, when the router buffer size (*limit*) is small and the buffer is the constraint, GSAM performs best, followed by LISA benefit from the reduced RTT and thus reduced retransmissions as analyzed above.

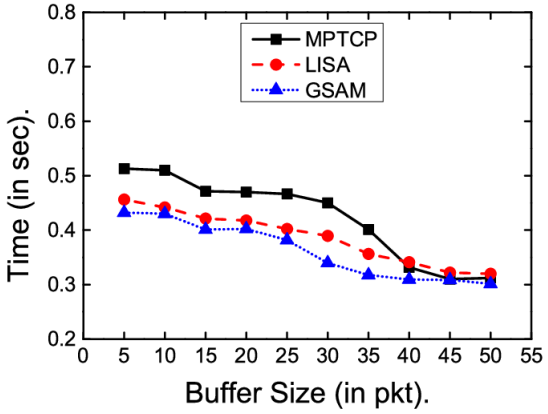


FIGURE 29. The flow completion of each algorithm with Codel under different *limit* values when the flow size is 300KB and the number of concurrent flows is 20.

Moreover, with the increasing buffer size, the difference between each algorithm becomes smaller and LISA performs slightly worse than MPTCP due to the slow increase of cwnd.

Above all, we can see that when the hard router buffer size (*limit*) is small, the router buffer overflows and thus retransmissions caused by the aggressive slow start behavior still exist with Codel. This is the issue what GSAM aims to alleviate and the results demonstrate that GSAM can achieve its goal by slowing down the window growth.

VI. CONCLUSIONS

MPTCP's slow start scheme over each subflow is uncoupled and behaves independently as regular TCP, causing congestion collapse and throughput reduction when these subflows share a common bottleneck. In this paper, we conduct an in-depth study to find the root reasons and propose GSAM to alleviate this issue. Based on each path's status information, GSAM firstly derives the appropriate threshold for smoothing the congestion window growth theoretically. Then, it smoothes the aggressive increasing of congestion window in slow start phase of MPTCP to avoid the buffer overflow which causes packet losses and even timeout events. Our Linux testbed results show that GSAM improves the MPTCP transfer completion time by reducing the number of retransmission and RTOs.

REFERENCES

[1] J. Zhao, J. Liu, H. Wang, and C. Xu, "Multipath TCP for datacenters: From energy efficiency perspective," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, May 2017, pp. 1–9.

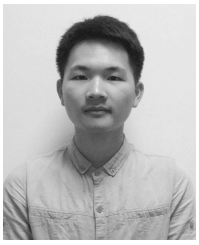
[2] P. Houzé, E. Mory, G. Texier, and G. Simon, "Applicative-layer multipath for low-latency adaptive live streaming," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7.

[3] J. Wu, B. Cheng, M. Wang, and J. Chen, "Quality-aware energy optimization in wireless video communication with multipath TCP," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2701–2718, Oct. 2017.

[4] Y. Zhang, H. Mekky, Z.-L. Zhang, F. Hao, S. Mukherjee, and T. V. Lakshman, "SAMPO: Online subflow association for multipath TCP with partial flow records," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.

[5] Q. De Coninck and O. Bonaventure, "Tuning multipath TCP for interactive applications on smartphones," in *Proc. IFIP Netw.*, 2018, pp. 514–522.
[6] S. R. Pokhrel, M. Panda, and H. L. Vu, "Analytical modeling of multipath TCP over last-mile wireless," *IEEE/ACM Trans. Netw.*, vol. 25, no. 3, pp. 1876–1891, Jun. 2017.
[7] J. Hwang, A. Walid, and J. Yoo, "Fast coupled retransmission for multipath TCP in data center networks," *IEEE Syst. J.*, vol. 12, no. 1, pp. 1056–1059, Mar. 2018.
[8] F. Duchene and O. Bonaventure, "Making multipath TCP friendlier to load balancers and anycast," in *Proc. IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2017, pp. 1–10.
[9] M. Kheirkhah, I. Wakeman, and G. Parisis, "MMPTCP: A multipath transport protocol for data centers," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.
[10] S. Ferlin, O. Alay, T. Dreibholz, D. A. Hayes, and M. Welzl, "Revisiting congestion control for multipath TCP with shared bottleneck detection," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.
[11] C. Raiciu, M. Handly, and D. Wischik, *Coupled Congestion Control for Multipath Transport Protocols*, document RFC 6356, Internet Eng. Task Force, 2011.
[12] C. Raiciu *et al.*, "How hard can it be? Designing and implementing a deployable multipath TCP," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, Berkeley, CA, USA: USENIX Association, 2012, p. 29.
[13] Q. Peng, A. Walid, and S. H. Low, "Multipath TCP algorithms: Theory and design," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 41, no. 1, pp. 305–316, 2013.
[14] R. Barik, M. Welzl, S. Ferlin, and O. Alay, "LISA: A linked slow-start algorithm for MPTCP," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7.
[15] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, *TCP Extensions for Multipath Operation With Multiple Addresses*, document RFC 6824, Internet Eng. Task Force, 2013.
[16] Y. Wang, K. Xue, H. Yue, J. Han, Q. Xu, and P. Hong, "Coupled slow-start: Improving the efficiency and friendliness of MPTCP's slow-start," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2017, pp. 1–6.
[17] S. Ferlin, Ö. Alay, O. Mehani, and R. Boreli, "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks," in *Proc. IEEE IFIP Netw.*, May 2016, pp. 431–439.
[18] F. Yang, Q. Wang, and P. D. Amer, "Out-of-order transmission for in-order arrival scheduling for multipath TCP," in *Proc. 28th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, May 2014, pp. 749–752.
[19] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent delay-aware packet scheduling for multipath transport," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 1222–1227.
[20] F. H. Mirani, N. Boukhatem, and M. A. Tran, "A data-scheduling mechanism for multi-homed mobile terminals with disparate link latencies," in *Proc. IEEE 72nd Veh. Technol. Conf.-Fall*, Sep. 2010, pp. 1–5.
[21] D. Ni, K. Xue, P. Hong, and S. Shen, "Fine-grained forward prediction based dynamic packet scheduling mechanism for multipath TCP in lossy networks," in *Proc. 23rd Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2014, pp. 1–7.
[22] D. Ni, K. Xue, P. Hong, H. Zhang, and H. Lu, "OCPS: Offset Compensation based Packet Scheduling mechanism for multipath TCP," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 6187–6192.
[23] P. Dong *et al.*, "Reducing transport latency for short flows with multipath TCP," *J. Netw. Comput. Appl.*, vol. 108, pp. 20–36, Apr. 2018.
[24] J. Han, K. Xue, H. Yue, P. Hong, N. Yu, and F. Li, "Receive buffer pre-division based flow control for MPTCP," *Mobile Ad-Hoc Sensor Netw.*, vol. 747, pp. 19–31, Mar. 2018.
[25] K. Xue *et al.*, "DPSAF: Forward prediction based dynamic packet scheduling and adjusting with feedback for multipath TCP in lossy heterogeneous networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 2, pp. 1521–1534, Feb. 2018.
[26] Y. Cao, M. Xu, X. Fu, and E. Dong, "Explicit multipath congestion control for data center networks," in *Proc. 9th ACM Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2013, pp. 73–84.
[27] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. 8th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2011, p. 8.
[28] R. Khalili, N. Gast, M. Popovic, and J.-Y. Le Boudec, "MPTCP is not Pareto-optimal: Performance issues and a possible solution," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1651–1665, Oct. 2013.
[29] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath TCP: Analysis, design, and implementation," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 596–609, Feb. 2016.
[30] P. Dong, J. Wang, J. Huang, H. Wang, and G. Min, "Performance enhancement of multipath TCP for wireless communications with multiple radio interfaces," *IEEE Trans. Commun.*, vol. 64, no. 8, pp. 3456–3466, Aug. 2016.
[31] Y. Cao, M. Xu, and X. Fu, "Delay-based congestion control for multipath TCP," in *Proc. 20th IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct./Nov. 2012, pp. 1–10.
[32] K. Nichols, V. Jacobson, A. McGregor, and J. Iyengar, *Controlled Delay Active Queue Management*, document RFC 8289, 2018.

PINGPING DONG received the B.S., M.S., and Ph.D. degrees from the School of Information Science and Engineering, Central South University, China. She is currently a Teacher with the College of Information Science and Engineering, Hunan Normal University, Changsha, China. Her research interests include protocol optimization and protocol design in wide area networks and wireless local area networks.



WENJUN YANG is currently pursuing the master's degree with the Department of Computer Education, Hunan Normal University, Changsha, China. His current research interests include protocol optimization for heterogeneous networks.

KAIPING XUE (M'09–SM'15) received the B.S. degree from the Department of Information Security, University of Science and Technology of China (USTC), in 2003, and the Ph.D. degree from the Department of Electronic Engineering and Information Science (EEIS), USTC, in 2007. From 2012 to 2013, he was a Postdoctoral Researcher with the Department of Electrical and Computer Engineering, University of Florida. He is currently an Associate Professor with the Department of Information Security and the Department of EEIS, USTC. His research interests include next-generation Internet, distributed networks, and network security.

WENSHENG TANG received the B.S. degree from Hunan Normal University, Changsha, China, in 1992, and the M.S. and Ph.D. degrees from the National University of Defense Technology, Changsha, in 1997 and 2009, respectively. He is currently a Professor with Hunan Normal University. His research interests include protocol optimization and cloud computing.

KAI GAO received the B.S. and Ph.D. degrees from Central South University, Changsha, China, in 2008 and 2014, respectively. He joined the Changsha University of Science and Technology, in 2015. His research interests include intelligent transportation systems and the Internet of Vehicles.

JIAWEI HUANG received the bachelor's degree from the School of Computer Science, Hunan University, in 1999, and the master's and Ph.D. degrees from the School of Information Science and Engineering, Central South University, China. He is currently a Professor with the School of Information Science and Engineering, Central South University. His research interests include performance modeling, analysis, and optimization for wireless networks and data center networks.