Shared Bottleneck-Based Congestion Control and Packet Scheduling for Multipath TCP

Wenjia Wei, Kaiping Xue, *Senior Member, IEEE*, Jiangping Han,
David S. L. Wei, *Senior Member, IEEE*, and Peilin Hong

***Abstract*— In order to be TCP-friendly, the original Multipath TCP (MPTCP) congestion control algorithm is always restricted to gain no better throughput than a traditional single-path TCP on the best path. However, it is unable to maximize the throughput over all available paths when they do not go through a shared bottleneck. Also, bottleneck fairness based solutions detect the bottleneck and conduct different congestion control algorithms at different bottleneck sets to increase throughput while remaining fair to single TCP. However, existing solutions generally detect shared bottlenecks through delay correlation and loss correlation between two flows, which often lead to misjudgement in dynamic and complex network scenarios. Therefore, in this paper, we first propose a new Shared Bottleneck based Congestion Control scheme, called SB-CC, which leverages ECN (Explicit Congestion Notification) mechanism to detect shared bottlenecks among subflows and estimate the congestion degree of each subflow. Then, with the congestion degree, SB-CC balances the loads among all subflows, and smooths out congestion window fluctuation. Also, in order to prevent throughput degradation due to out-of-order packets, we propose a Shared Bottleneck based Forward Prediction packet Scheduling scheme, called SB-FPS. SB-FPS distributes data according to the window size changes of each subflow, and thus could more accurately schedule data in shared bottleneck scenarios. We implement our proposed scheme in the Linux kernel and simulation platform to evaluate the performance in different scenarios. Measurement results indicate that our scheme can detect the bottleneck more accurately and improve the overall network performance while still keeping bottleneck fairness.**

***Index Terms*— TCP-friendly, multipath TCP, shared bottleneck, explicit congestion notification (ECN), congestion control, packet scheduling.**

I. Introduction

NOWADAYS, it's not unusual that an electronic device is equipped with more than one network interface, *e.g.*, a mobile phone with 3G/4G and WiFi interfaces. The concurrent use of multiple interfaces will bring a better Quality of Service (QoS) for high bandwidth consumption applications such as multimedia streaming and real-time games [1], [2]. Unfortunately the use of multiple interfaces is not supported by legacy TCP [3]. As an extension of TCP, MultiPath TCP (MPTCP) [4] has already attracted much attention due to its capability of using multiple interfaces concurrently and compatibility with TCP. MPTCP splits data and transmits them simultaneously through multiple interfaces. Each path over one pair of interfaces of two MPTCP terminals is defined as a *subflow*. The implementation of MPTCP only requires the modification of the transport layer, which is transparent to both application layer and network layer.

Congestion control is a key component of TCP, which controls the amount of traffic transmitted over a connection and avoids congestion. The basic principles of MPTCP congestion control are "improve throughput" and "do no harm" [5]. This means: 1) When there is no congestion, an MPTCP connection should achieve higher throughput than a single TCP connection; 2) When the congestion occurs, MPTCP should be friendly to single-path TCP, which means that the throughput achieved are nearly the same. Various congestion control mechanisms, such as LIA [6], OLIA [7], wVegas [8], BALIA [9], and Couple+ [10], have been proposed under these principles. However, all these schemes are designed to achieve network fairness, which means the overall throughput of an MPTCP connection should be no higher than that of a single TCP on the best end-to-end path no matter whether MPTCP subflows share a bottleneck or not. Although network fairness makes all connections achieve the same throughput to ensure fairness, it actually is unfair to MPTCP users. Besides, network fairness based coupled congestion control algorithm limits subflows' performance at non-shared-bottleneck links (*i.e.*, the links other than the shared bottleneck), and it only achieves the same transmission performance as legacy TCP users.

Considering the unfairness to MPTCP users in *network fairness* based schemes, Dynamic Window Coupling (DWC) [11], a *bottleneck fairness* based approach, was proposed. DWC restricts the total throughput of the subflows sharing a bottleneck to be no higher than that of a single-path TCP at this bottleneck, while other subflows passing through different bottlenecks act as individual TCPs. *Bottleneck fairness* is friendly with traditional TCP connections, and can improve the throughput of the subflows on the disjoint paths if the congestion control schemes can accurately detect the bottlenecks. However, in DWC [11], the proposed schemes of

Manuscript received May 22, 2019; accepted January 11, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Kuzmanovic. Date of publication February 13, 2020; date of current version April 16, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61972371 and Grant 61671420 and in part by the Youth Innovation Promotion Association of the Chinese Academy of Sciences (CAS) under Grant 2016394. *(Corresponding author: Kaiping Xue.)*

Wenjia Wei, Kaiping Xue, Jiangping Han, and Peilin Hong are with the Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei 230027, China (e-mail: kpxue@ustc.edu.cn).

David S. L. Wei is with the Computer and Information Science Department, Fordham University, Bronx, NY 10458 USA.

Digital Object Identifier 10.1109/TNET.2020.2970032

delay correlation and loss correlation sometimes fail to reveal real shared bottleneck situations and thus cause MPTCP to have poor transmission performance. For example, random packet loss often occurs in lossy networks and the increase of path delay is usually caused by load increase on non-shared bottleneck links.

Instead of utilizing packet loss and delay signals, some explicit network feedback based schemes, such as [12] and [13], have been proposed to obtain the real-time network situation to further improve the network performance. In this paper, based on ECN (Explicit Congestion Notification) signal [14] which can accurately reflect the network congestion situation, we propose a newly designed Shared Bottleneck based Congestion Control scheme, called SB-CC. SB-CC consists of a new shared bottleneck detection algorithm and a newly designed bottleneck fairness based congestion control algorithm. The shared bottleneck detection scheme utilizes ECN signal to detect shared bottlenecks for subflows belonging to a MPTCP connection. ECN-enabled devices have existed in Ubuntu Linux since 12.04 and in Windows Server since 2012 [15]. Proportion of the most popular websites that support for ECN has increased from 8.5% in 2012 to over 70% in May 2017 [15]. Cisco IOS routers perform ECN marking if configured with the WRED queuing discipline since version 12.2(8)T, and more and more newly manufactured Cisco routers support ECN mechanisms [16]. In the ECN mechanism, when the queue length exceeds the marking threshold at a specific intermediate router, forwarded packets will be marked at a certain probability. Then, subflows receiving marked ACKs in the same time period can be judged to share a bottleneck. Meanwhile, the sender implements the judgement process twice to make sure that two judgment decisions are consistent so as to avoid causing mis-judgement and to ensure the detection accuracy. Compared to delay signal and loss signal, ECN signal can explicitly reflect the actual congestion status in network more accurately. Based on the proposed shared bottleneck detection scheme, subflows in a MPTCP connection passing a shared bottleneck are grouped into a set to implement coupled congestion control. Therefore, this set of MPTCP subflows can be treated as a whole and should be friendly with the concurrent regular TCP flows sharing the same bottleneck. Our proposed coupled congestion control algorithm is based on the defined subflow congestion degree, which can be used to balance the load among the different MPTCP subflows by considering bottleneck fairness. In the proposed coupled congestion control mechanism, we modify the "half window" operation of the legacy MPTCP, and elastically control the window increment and decrement degree according to the subflow congestion degree.

Obviously, the proposed SB-CC algorithm can detect shared bottlenecks more accurately and improve the performance of MPTCP while providing bottleneck fairness. However, with the original scheduling algorithm for MPTCP, the throughput will decrease in heterogeneous network scenarios, since the asymmetry of delay and bandwidth in different subflows will result in a large number of out-of-order packets and further cause receive buffer blocking [17]. The maximum blocking time will increase as the diversity of link delay and bandwidth increases among different subflows [18]. Some intelligent scheduling algorithms, such as BLEST [19], F^2P-DPS [20] and DSPAF [21], have been proposed to reduce the possibility of out-of-order packets from different aspects. However, all these scheduling algorithms only treat each subflow as a single TCP and did not consider that all subflows' congestion windows should be coupled in the congestion avoidance phase to ensure fairness. As a result, they do not perform well in shared bottleneck scenarios. To solve this problem, we further develop a fine-grained Shared Bottleneck based Forward Prediction packet Scheduling scheme, named SB-FPS, for MPTCP. SB-FPS takes detected shared-bottleneck sets into consideration, model each subflow's future behavior and pre-schedule data transmission in fine-grained control.

Furthermore, we realize the proposed scheme in Linux kernel and construct realistic experiment scenarios to verify the performance improvement of of our proposed scheme over the existing schemes.

The main contributions of our work can be summarized as follows:

- We propose a new Shared Bottleneck based coupled Congestion Control scheme, called SB-CC. SB-CC consists of a newly designed shared bottleneck detection algorithm and a newly designed bottleneck fairness based coupled congestion control algorithm by taking congestion degree into consideration. We leverage the ECN mechanism to explicitly expose subflows sharing one or more bottlenecks in an MPTCP connection. The judgement process should be implemented twice to make sure the consistence of the two judgement decisions so as to ensure the detection accuracy. Further, for providing load balance while keeping bottleneck fairness, the proposed coupled congestion control algorithm estimates the congestion degree of each subflow and utilize it to elastically control window increase and decrease.
- We further develop a fine-grained Shared Bottleneck based Forward Prediction packet Scheduling scheme (SB-FPS) for MPTCP. For each subflow, by taking the detected shared bottleneck sets into consideration, we model each subflow's future behavior and pre-schedule data transmission in fine-grained control in shared bottleneck scenarios.
- By implementing the proposed scheme and some existing schemes to be compared in Linux kernel and simulation platform, we conduct realistic experiments in different scenarios. The evaluation results show that our scheme can detect the shared bottleneck more accurately and quickly, and the algorithms of coupled congestion control and pre-scheduling can further improve the overall throughput while keeping bottleneck fairness.

The rest of this paper is organized as follows. In Section II, we introduce the existing congestion control algorithms, shared bottleneck detection scheme, and packet scheduling algorithms in MPTCP. A detailed description of our proposed scheme is provided in Section III. In Section IV, we give the experiment results and performance analysis. Finally, the conclusion is given in Section V.

II. Related Work

In this section, we introduce the related works on congestion control, shared Bottleneck detection, and packet scheduling in TCP/MPTCP.

A. Congestion Control in MPTCP

The most common congestion control mechanism in TCP, is TCP NewReno [3], which is based on AIMD (Additive Increase Multiplicative Decrease) [22]. The default TCP Cubic in current Linux kernel also inherits this basic mechanism. In MPTCP, if each subflow in a MPTCP connection runs TCP NewReno independently, the MPTCP connection will take much more resources than a single-path TCP flow sharing a bottleneck, which is not friendly for traditional TCP connections. In order to be TCP-friendly, some MPTCP congestion control algorithms, such as EWTCP [23], LIA [6], OLIA [7], wVegas [8], and BALIA [9], adopt semi-coupled approach to dynamically adjust the congestion window size. However, they are all based on TCP NewReno and only modify the congestion avoidance phase to perform coupled congestion control for all subflows. The increase rate of aggregate congestion window for a MPTCP connection is the same as that of a single-path TCP to ensure that the MPTCP connection consumes no more resources than a single-path TCP, which achieves network fairness.

All these schemes simply couple all subflows to achieve network fairness, which do not rely on shared bottleneck detection mechanisms. If shared bottleneck sets of subflows can be detected, MPTCP can control subflows sharing the same bottleneck to gain aggregate throughput no higher than a regular TCP flow, while imposing no restrictions to subflows in non-shared-bottleneck paths, and different shared bottleneck sets can act like single-path TCP flows from the respective shared bottlenecks. Therefore, such a scheme can satisfy the principles of "do no harm" and "improve throughput" at the same time, provided the shared bottleneck sets need to be timely and successfully detected first.

B. Shared Bottleneck Detection

TCP is an end-to-end transmission protocol, which is transparent to intermediate routers. Based on this principle, it is difficult to detect the shared bottlenecks. DWC [11] is the first scheme for MPTCP to achieve bottleneck fairness, where the subflows which encounter delay increase or packet loss during the same time period will be considered to share the same bottleneck. In our previous work [24], presented and realized a shared bottleneck detection scheme based on congestion interval variance measurement, which is robust and effective to the path lag problem. Ferlin *et al.* [25] used three key statistics of one way delay to detect shared bottleneck sets. However, this scheme needs a long time to obtain the detection result. Meanwhile, it must modify the MPTCP Time Stamp option to feed back the bottleneck congestion information on each subflow, which requires some changes to the existing network architecture and protocol stack. Besides, the background traffic on non-bottleneck links usually leads to the decrease of the bottleneck detection accuracy.

There are also some efforts devoted to detect the existence of a bottleneck shared between two legacy TCP flows. Zhang *et al.* [26] leveraged loss correlation to predict TCP flows that traverse through a shared bottleneck. Kim *et al.* [27] proposed a wavelet-based technique that uses a signal processing method, named wavelet denoising, to detect whether TCP flows share a bottleneck. Yousaf and Welzl [28] used path delay correlation on two TCP flows to detect whether they share a bottleneck.

Since random packet loss often occurs in lossy networks and the path delay increase is usually caused by load increase on non-shared bottleneck links, the shared bottleneck sets will likely be misjudged in the above mentioned schemes.

C. Packet Scheduling in MPTCP

MPTCP establishes a connection with multiple subflows on different paths. In MPTCP layer, data, in the form of packets, are scheduled to parallel subflows for transmission. Round-Robin is the simplest scheduling algorithm for MPTCP, in which all subflows have the same priority and the sender just schedules data from the sending buffer in sequence to the available sending windows of all subflows in the order of polling. It is unable to alleviate the effects caused by heterogeneous path characteristics, leading to serious problem of out-of-order packets, where packets with larger sequence number may arrive at receiver earlier than packets with smaller sequence number, and have to wait until the arriving of the packets with smaller sequence numbers. In current MPTCP specification [4], Lowest RTT First (LowRTT) is used as the default scheduling algorithm, which relies on the round-trip time (RTT) measured on each subflow and sends segments over the subflow having the lowest round-trip time first. However, LowRTT also fails to guarantee packets' arriving at the receiver in order. In-order delivery among different paths remains a main challenge for multipath transmission, where out-of-order packets will cause the head-of-line blocking problem at receiver [29]. Moreover, this problem will become more serious in asymmetric scenarios, where multiple paths have significantly different end-to-end delay. Therefore, usually, MPTCP scheduling algorithms introduce a large receive buffer shared by all subflows to hold more out-of-order packets at receiver so as to reduce the impact of out-of-order packets. However, using large receive buffer is not able to provide performance improvement for real-time data transmission.

Furthermore, some intelligent scheduling algorithms have been proposed to make arriving packets in order as much as possible. Linux-MPTCP scheduler [30] is the first practical scheduler implemented in Linux MPTCP kernel [31], which frequently allocates data to all subflows in proportion according to the current congestion window size and path delay on each path. Mirani *et al.* [32] proposed Forward Prediction Packet Scheduling (FPS) to reduce the number of out-of-order packets due to asymmetric path delay on different paths. This introduces fine-grained prediction about data transmission duration and congestion window adjustment of several rounds on the faster subflows. Different from FPS, our proposed scheme [21] is a newly designed fine-grained

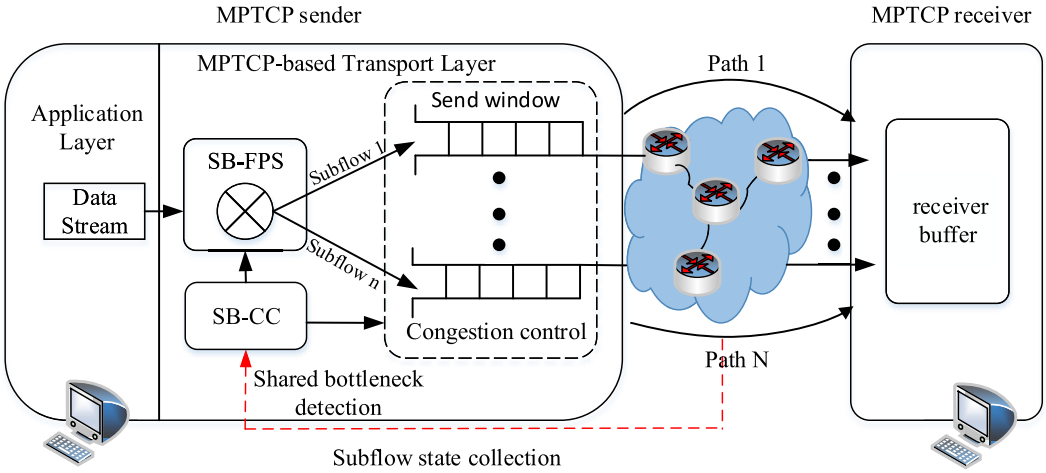


Fig. 1. Framework of the proposed scheme. ① SB-CC groups the subflows into different bottleneck set according to the state of each subflow. The subflows in the same bottleneck are coupled controlled. ②Based on the information provided by SB-CC, SB-FPS models each subflow's behavior and estimates the number of packets scheduled to each subflow.

forward prediction based dynamic packet scheduling scheme, in which both of the random packet loss and time offset are considered. Therefore, compared to FPS, our scheme does better in lossy network scenarios.

D. Summary

The above schemes about packet scheduling all treat a subflow as a single TCP and did not consider implementing coupled congestion control, so in congestion avoidance phase, the congestion window (CWND) in each subflow grows by one for every RTT. However, the evolution of CWND on different subflows in one shared bottleneck set should be correlated with each other, which means that the CWND will not simply grow by one for every RTT. In our scheme, we introduce the detection of shared bottleneck sets, based on the consideration of bottleneck fairness for MPTCP. We further model the increase and decrease of each subflow's CWND, and design a reasonable coupled congestion control algorithm and a fine-grained scheduling algorithm. In order to more accurately estimate the number of packets supposed to be scheduled to each subflow, our scheme integrately considers both the congestion control and the packet scheduling.

III. Shared Bottleneck Based Congestion Control and Packet Scheduling for Multipath TCP

In this section, we describe our proposed scheme that combines congestion control and packet scheduling to improve the performance of MPTCP, while keeping bottleneck fairness. As shown in Fig. 1, the proposed scheme consists of two main parts, namely SB-CC and SB-FPS. SB-CC first leverages ECN mechanism to divide different subflows into different bottleneck sets. The different subflows that receive ECE-marked ACKs at the same time period will be judged as sharing the same bottleneck. Meanwhile, the sender implements the judgement with two stages and makes sure that two judgement decisions are consistent so as to ensure a high detection accuracy. Then, SB-CC groups subflows in the same shared bottleneck set to implement subflow congestion degree based coupled congestion control. SB-FPS is a new intelligent bottleneck fairness based packet scheduling algorithm that models each subflow's CWND according to its congestion degree and the relationship among subflows whether they are in the same shared bottleneck set. Therefore, the number of packets to be transmitted on each subflow can be estimated and scheduled more effectively.

A. Shared Bottleneck Based Congestion Control

1) Our Proposed Shared Bottleneck Detection Scheme: We utilize the ECN mechanism in the design of shared bottleneck detection for MPTCP to get more accurate detection results. According to [33] and other configuration values, we set a threshold K in ECN-enabled routers where passing packets are marked "11" in ECN field when the average queue length in an intermediate router exceeds the threshold. Then, when receiving ECN marked TCP segment, the receiver sets the ECE (ECN-Echo flag) with the value "1" in its responsed ACK to notify the sender. The sender will then know the congestion state in the corresponding subflow. Furthermore, the congestion degree of each subflow can be obtained from the statistical analysis on the proportion of marked TCP segments within a sending window, which can be updated at the end of each sending window. Our proposed shared bottleneck detection scheme involves two stages of judgement.

We introduce a function module of shared bottleneck detection to monitor all subflows for a MPTCP connection. At the very beginning, when there are one or more ECE-marked ACKs detected in a subflow (*e.g.*, $subflow_1$), our scheme will trigger first for this subflow and launch the detection of ECE marked ACKs and packet loss signals in all other subflows within the predefined observation window. The observation window, as shown in Fig. 2, consists of the past and the future observation window, and the size of both windows is $w_r/2$ [11], where w_r is the congestion window size of subflow r.

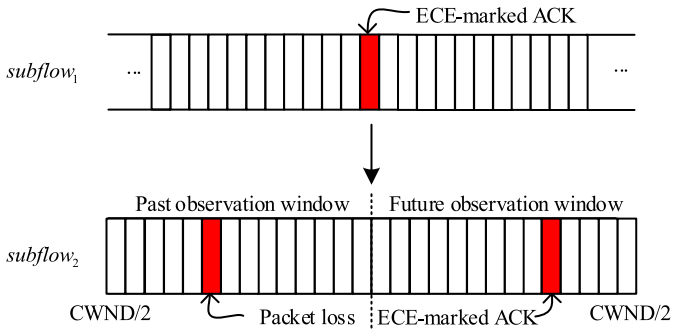


Fig. 2. When *subflow*$_1$ receives an ECE-marked ACK, *subflow*$_2$ checks the past and future observation window for packet loss and ECE-marked ACK.

If the ECE-marked ACKs or packet loss signals are detected on one subflow (*e.g.*, *subflow*$_2$) during the observation window, *subflow*$_1$ and *subflow*$_2$ will be grouped into the "Preliminary Shared Bottleneck Set" and the second stage are launched to verify this judgement result. Meanwhile, other subflows that do not belong to any sets still stay in the Monitoring state and repeat the above process. To be noted, the priority we consider is ECN signal, but we also consider packet loss signal. If there's no returned ACK for a transmitted TCP segment in a specific subflow until the set re-transmission timer is time-out, it indicates that a congestion event in this subflow has happened in the observation window.

The second stage is conducted to verify whether the preliminary shared bottleneck set is a correct one. The sender will monitor all the subflows in this preliminary shared bottleneck set to firstly detect one or more ECE-marked ACKs in a subflow. Then, the sender will judge whether congestion events within the observation window also occurs in all other subflows in this preliminary shared bottleneck set but not in other subflows that are still in monitoring state outside this preliminary shared bottleneck set. If so, preliminary shared bottleneck set is verified and move this set to the Final Judgement state. If not, move all the subflows in this set back to the Monitoring state.

Pseudo-code of our proposed shared bottleneck detection algorithm is given in **Algorithm 1**. For subflows in a set in the Final Judgement state, the sender always observe all these subflows and implement the above verification process. If the shared bottleneck set is still correct, keep it and stay in the Final Judgement state. If it's not correct, move all the subflows in the set back to the Monitoring state.

For clarity, the state-transition diagram of shared bottleneck detection for all subflows in a MPTCP connection is shown in Fig. 3.

2) Bottleneck Fairness Based Coupled Congestion Control: In SB-CC, for window reduction part, we define congestion degree, the concept similar to that in DCTCP, and use fine-grained window reduction similar to that in DCTCP for each MPTCP subflow. Then, to achieve bottleneck fairness and keeping stable, we also make use of the defined congestion degree in constrained additive increase so as to provide coupled fine-grained bottleneck-fairness congestion control. Furthermore, to be noted, our scheme refers to RTT-Compensator, which is still a network-fairness-based congestion control

Algorithm 1 Shared Bottleneck Detection Algorithm

Input: Denote R as the set of subflows in a MPTCP connection and each subflow $r \in R$, observation window of sublow r, w_r, state of subflow r, $state[r] \in \{monitor, judgment\}$;

Output: Shared bottleneck set bs_i, i represents the ith shared bottleneck set;

```
Initialization: state[r]=monitor, i = 0;
foreach subflow in a MPTCP connection do
    /* monitor all subflows' state and congestion events*/
    if subflow r* ∈ R receive ECE then
        if state[r*]=monitor then
            foreach subflow r ∈ R\r* do
                if subflow r receive ECE or packet loss within the observation window w_r then
                    state[r*]=judgement;
                    state[r]=judgement;
                    subflow r*∈ bs_i;
                    subflow r∈ bs_i;
            /* verify the Preliminary Shared Bottleneck Set */
            if subflow r'∈bs_i receive ECE then
                if (each subflow r ∈ bs_i\r' receive ECE or packet loss within w_r) && (any subflow r ∉ bs_i do not receive ECE or packet loss within w_r) then
                    Return bs_i;
                    i++;
                else
                    foreach subflow r∈bs_i do
                        state[r]=monitor;
Return bs_i;
```

scheme. However, our proposed SB-CC is a bottleneck-fairness-based coupled congestion control scheme that can provide higher throughput performance than existing MPTCP congestion control algorithms while still keeping bottleneck fairness. From the perspective of data distribution, dynamic adjustment and load balance can be made between different MPTCP subflows. The details of our proposed coupled congestion control are described as follows.

Dynamically Multiplicative Decrease: Different from legacy TCP and DWC, SB-CC takes the subflow congestion degree β_r into consideration to dynamically decrease the congestion window. We refer to the decrease phase in DCTCP [12], which introduces the concept of congestion degree. Referring to DCTCP [12], we define the proportion of packets that are marked by ECN mechanism within each window as the congestion degree of *subflow*$_r$, and it should be updated for each sending windows as:

$$\beta_r = g \cdot T_r + (1 - g) \cdot \beta_r, \tag{1}$$

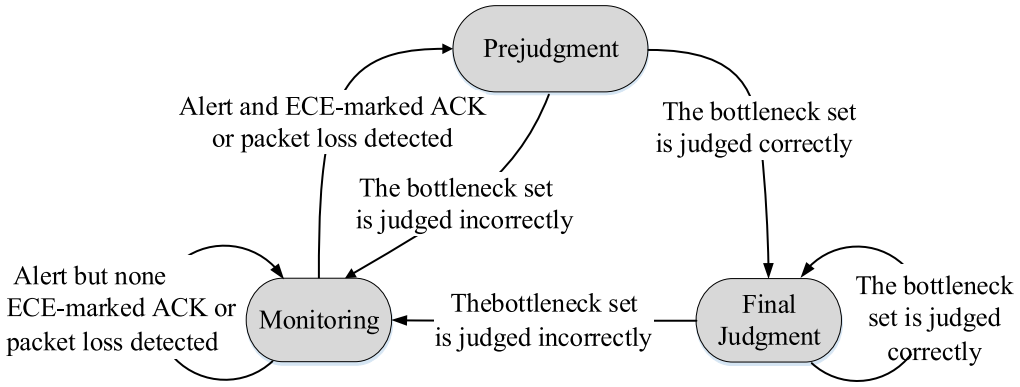


Fig. 3. State-transition diagram of the shared bottleneck detection process.

where T_r is the proportion of packets that were marked in the last sending window of packets, g is the weight given to T_r, and we set $g = 1/8$ (It's an empirical value, which refers to the smoothed value used for the average RTT measurement.). When a subflow (*e.g.*, *subflow*$_r$) receives an ECE marked ACK or detects the event that the set re-transmission timer is time-out, it can indicate that a congestion event in this subflow has occured, and the window size should be changed to:

$$w_r = (1 - \frac{\beta_r}{2}) \cdot w_r, \tag{2}$$

where w_r is the congestion window size of *subflow*$_r$. When β_r is close to 1, it means *subflow*$_r$ suffers severe congestion and the situation degrades to legacy TCP. On the contrary, when β_r is close to 0, it means that there is less congestion in *subflow*$_r$, and the congestion window only needs to decrease slightly.

Constrained Additive Increase: For bottleneck fairness based coupled congestion control, when the congestion window decrease strategy is changed, the increase strategy should be updated accordingly. According to the features of bottleneck fairness, the subflows grouped into a shared bottleneck set should achieve as nearly equal throughput as that of a single-path TCP that shares the same bottleneck [34], which can be represented as:

$$\sum_{r \in S} \frac{\hat{w_r}}{RTT_r} = \max_{r \in S} \frac{\hat{w_r}^{TCP}}{RTT_r}, \tag{3}$$

where $\hat{w_r}$ is the equilibrium congestion window size of subflow r, and $\hat{w_r}^{TCP}$ is the equilibrium window that would be obtained by a single-path TCP at the same bottleneck. In our algorithm, subflow $r \in S$ increases its window size w_r by $\min(a/w_{bs}, 1/w_r)$, where w_{bs} is the aggregate congestion window of bottleneck set S and the parameter a is the aggressiveness factor that controls the increment of subflow r's window. Therefore, finding an appropriate a is the key to achieving bottleneck fairness. For each subflow, the increase and decrease of the congestion window must be balanced out [35], [36], which can be arithmetically expressed in Eq. (4).

$$\min\left(\frac{a}{\hat{w_{bs}}}, \frac{1}{\hat{w_r}}\right)(1 - \lambda_r) = \frac{\beta_r}{2}\hat{w_r}\lambda_r, \tag{4}$$

Algorithm 2 SB-CC Algorithm

Input: RTT_r, subflow congestion degree β_r;
Output: Congestion window size w_r for each subflow r;

Initialization: $g = 1/8$, $\beta_r = 1$;
foreach *ack on subflow* r **do**
  Find the shared bottleneck set bs_i that subflow r belongs to by executing Algorithm 1.
  if *subflow* $r \in bs_i$ **then**
    $\beta_r \leftarrow g \cdot T_r + (1 - g) \cdot \beta_r$;
    $a \leftarrow \beta_r \hat{w_{bs_i}} \frac{\max_r \hat{w_r}/RTT_r^2}{(\sum_r \hat{w_r}/RTT_r)^2}$;
    $w_r \leftarrow w_r + min(\frac{a}{w_{bs_i}}, \frac{1}{w_r})$;
  if *subflow* r *does not belong to any* bs **then**
    $w_r \leftarrow w_r + \frac{1}{w_r}$;
foreach *loss or ECE on subflow* r **do**
  $w_r \leftarrow w_r - \frac{\beta_r}{2} \cdot w_r$;
return *congestion window size* w_r*;*

where λ_r is the congestion loss rate on subflow r. Furthermore, we can compute a from Eq. (4). Since λ_r is usually very small, *i.e.*, $1 - \lambda_r \approx 1$. From Eq. (3), Eq. (4), and $\hat{w_r}^{TCP} = \sqrt{2/\lambda_r}$, subflow r increases the window size w_r, and the corresponding aggressiveness factor a can be computed as:

$$\begin{aligned} w_r &= w_r + min(\frac{a}{w_{bs}}, \frac{1}{w_r}), \\ a &= \beta_r \hat{w_{bs}} \frac{\max_r \hat{w_r}/RTT_r^2}{\left(\sum_r \hat{w_r}/RTT_r\right)^2}. \end{aligned} \tag{5}$$

Algorithm SB-CC is shown in **Algorithm 2**. By jointly considering subflows in one bottleneck set, our proposed SB-CC can achieve the goal of "do no harm" to other TCP connections sharing the same bottleneck and meanwhile improve the overall network performance. To be noted, our proposed SB-CC can detect and maintain multiple bottleneck sets for one MPTCP connections, and it also adapts to the dynamic changing scenarios of shared bottlenecks.

B. Shared Bottleneck Based Forward Prediction Packet Scheduling

To mitigate the problems associated with the oversimplified packet transmission scheduling in MPTCP with default

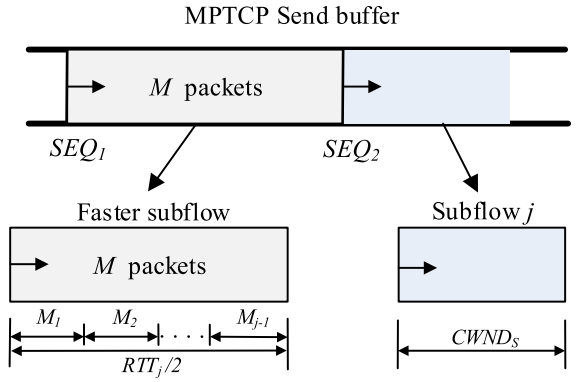


Fig. 4. Packet transmission process of SB-FPS.

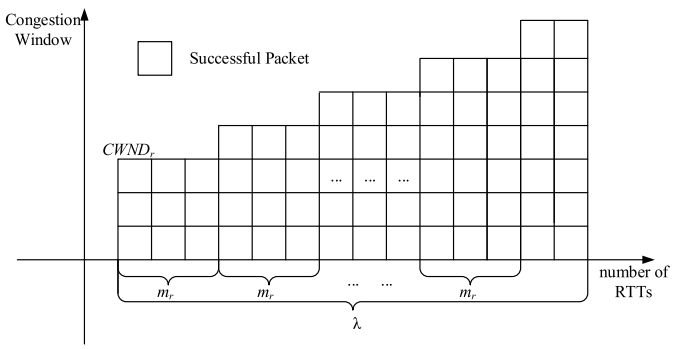


Fig. 5. The congestion avoidance phase in coupled congestion control.

LowRTT, the scheduling mechanism should reasonably allocate each packet on a specific path so as to ensure that packets arrive at the receiver in order. In this section, based on the detected shared bottleneck sets with SB-CC, we introduce a new shared bottleneck based forward prediction packet scheduling algorithm, called SB-FPS, to address the problem of out-of-order packets and unnecessary retransmissions caused by this. This type of scheduling can provide performance improvement, especially the real-time throughput, in heterogeneous networks.

For a subflow that is being scheduled, SB-FPS predicts the data amount that sent on other faster subflows with consideration of the detection result of shared bottleneck sets and further predicts the different growth rate of each subflow's CWND in different shared bottleneck sets. Then, the slower subflow gaps this data to send, so as to make all the data arrive at the receiver in order.

Assuming that an MPTCP connection has n subflows, which are arranged by RTT from small to large, *i.e.*, $RTT_1 \leq \cdots \leq RTT_j \leq \cdots \leq RTT_n$. When subflow j is ready to send new data and it is under-scheduling, SB-FPS predicts the number of packets M that can be transmitted by all the subflows whose RTT is less than RTT_j during $\frac{RTT_j}{2}$. As shown in Fig. 4, the sequence number scheduled on the faster subflows starts from SEQ_1, then subflow j gaps M packets in the sending buffer and transmits from the sequence number $SEQ_2 = SEQ_1 + M$ in the buffer to send until its congestion window is full. By following this method, the receiver can complete the reception of scheduled packets almost simultaneously on all subflows.

Let M_r be the number of packets that can be transmitted by each subflow r during $\frac{RTT_j}{2}$, the overall gap packets M can be: $M = \sum_1^{j-1} M_r$. To be noted, if $j = 1$, which means subflow j is the fastest subflow, then $M = 0$. So the most important thing is to estimate M_r of each subflow r.

As the CWND changes according to SB-CC, the growth rates of subflows in different shared bottleneck sets are different. Within the shared bottleneck set, the growth of windows is coupled together, the growth of each subflow r during each RTT is $w_r * \min(\frac{a}{w_{bs}}, \frac{1}{w_r})$ and less than one unit. Therefore, CWND of the subflow inside the shared bottleneck set needs serveral RTTs to grow by one unit, while the independent subflow only needs one RTT. SB-FPS accurately calculates the amount of packets M based on the SB-CC algorithm to make all of these packets arrive at the receiver in order. This is the main idea of packet allocation for SB-FPS. According to [36], as given in **Algorithm 3**, we calculate M_r in the case of subflow r in and outside of the bottleneck set, when subflow j is under-scheduling.

1) Subflow r is in the Bottleneck Set bs: When the sender receives an ACK on subflow r, the increase of CWND during the congestion avoidance phase is shown in Eq. (5). Thus, the increase of subflow $r's$ CWND after one RTT without packet loss will be

$$w_r * \min(\frac{a}{w_{bs}}, \frac{1}{w_r}) = \min(\frac{a * w_r}{w_{bs}}, 1). \tag{6}$$

Since the growth of the subflow inside the bottleneck set during one RTT is smaller than one unit, the subflow requires a certain number of RTTs to increase its CWND by one unit. Fig. 5 shows the packet transmission process in the congestion avoidance phase on subflow r. A column represents the packets sent during one RTT. m_r RTTs are needed for subflow $r's$ CWND to be increased by one unit and m_r can be calculated by Eq. (7):

$$m_r = \max(\frac{w_{bs}}{a * w_r}, 1), \quad a = \beta_r \hat{w_{bs}} \frac{\max_r \hat{w_r}/RTT_r^2}{\left(\sum_r \hat{w_r}/RTT_r\right)^2}. \tag{7}$$

During $\frac{RTT_j}{2}$, subflow r transmits $\lambda = \frac{RTT_j/2}{RTT_r}$ rounds of data and its CWND increases by one unit every m_r RTTs. So we can calculate the number of packets M_r that subflow r can transmit during $\frac{RTT_j}{2}$ as:

$$\begin{aligned} M_r &= \frac{\lambda/m_r}{2} * (2w_r + \lambda/m_r - 1) * m_r \\ &= \frac{\lambda}{2} * (2w_r + \lambda/m_r - 1). \end{aligned} \tag{8}$$

2) Subflow r is an Independent Subflow: When subflow r does not belong to any bottleneck set, it will act as a regular TCP flow. Then, its CWND during the congestion avoidance phase smoothly increases by one for each RTT. So $m_r = 1$. As a result, the number of packets M_r allocated to subflow r during the $\frac{RTT_j}{2}$ is:

$$M_r = \frac{\lambda}{2} * (2w_r + \lambda - 1). \tag{9}$$

Algorithm 3 SB-FPS Algorithm

Input: RTT_r, the aggressiveness factor a, congestion window size w_r;

Output: The data amount M allocated to the faster subflows;

Initialization: sorts all the subflows according to their estimated RTT values in non-decreasing order, $RTT_1 \leq \cdots \leq RTT_j \leq \cdots \leq RTT_n$;

foreach *subflow* j *under-scheduling* **do**

  foreach *subflow* $r \leqslant j$ **do**

    Find the shared bottleneck set bs that subflow r belongs to by executing Algorithm 1.

    if *subflow* $r \in bs_i$ **then**

      $\mathrm{E}[m_r] = \frac{w_{bs_i}}{a * w_r}$;

      $M_r \leftarrow \frac{\lambda}{2} * (2w_r + \lambda/\mathrm{E}[m_r] - 1)$;

    if *subflow* r *does not belong to any* bs **then**

      $M_r \leftarrow \frac{\lambda}{2} * (2w_r + \lambda - 1)$;

  $M \leftarrow \sum_1^{j-1} M_r$;

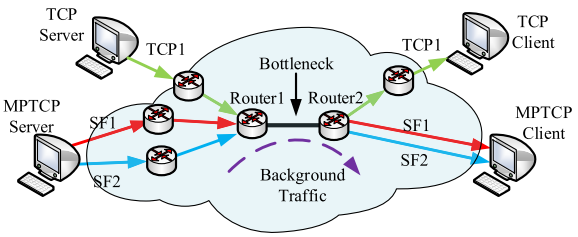
return M

IV. Performance Analysis

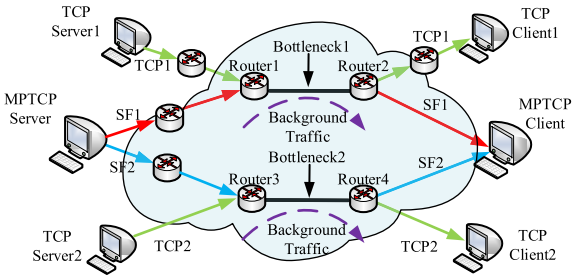
A. Performance Analysis of SB-CC

It is difficult to verify our technique directly on a real network, *e.g.*, Internet, because we could not know whether the two flows share a bottleneck or not to confirm the correctness of our technique. Through the constructed experimental scenarios are with 14 Linux hosts, we evaluate the performance of our SB-CC compared with DWC [11], SBD [25], BALIA [9] and OLIA [7] for scenarios as shown in Fig. 6. Besides, we add background traffic to create a more realistic environment. The majority of traffic ($> 90\%$) traverses the bottlenecks as the background traffic, which consists of both TCP and UDP flows with varied lengths generated by D-ITG [37]. We use both symmetric (both subflows' path delay are 60ms) and asymmetric (subflows' path delay are 60ms and 120ms, respectively) scenarios to verify whether SB-CC performs better and steadier than the compared algorithms. We have conducted 20 independent tests in each scenario and every test runs for 300 seconds. We take the average of multiple measurements as the final results.

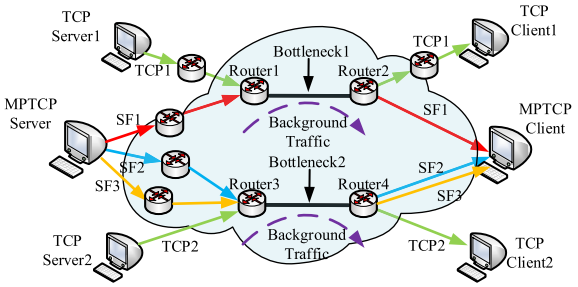
1) Window Fluctuation Performance: In section III-A, we have shown that SB-CC modifies the congestion control algorithm with subflow congestion degree. Therefore, SB-CC can dynamically adjust the congestion window according to the path quality and smooth out the window fluctuation. Now we use a simple simulation experiment to verify it. The experiment scenario is shown in Fig. 6(a). We run DWC, SBD, and SB-CC, respectively, with no background flows and monitor the congestion window size of two subflows and their aggregate window size. Fig. 7 shows the contrasts of congestion window fluctuation performance of DWC, SBD, and SB-CC, respectively. We can see that both congestion windows of DWC and SBD fluctuate more significantly than SB-CC. For multimedia applications, the highly fluctuating sending rate is undesirable. Compared with DWC and SBD, the range of window size variation in our scheme is less, and the overall average CWND size in SB-CC is larger than that in DWC, and our scheme is more suitable to support applications that require a steady throughput.



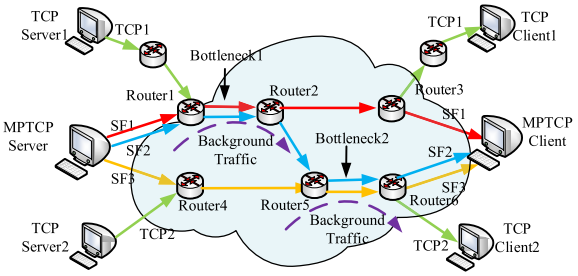
(a) Scenario 1: SF1 and SF2 share the same bottleneck



(b) Scenario 2: SF1 and SF2 traverse distinct bottlenecks



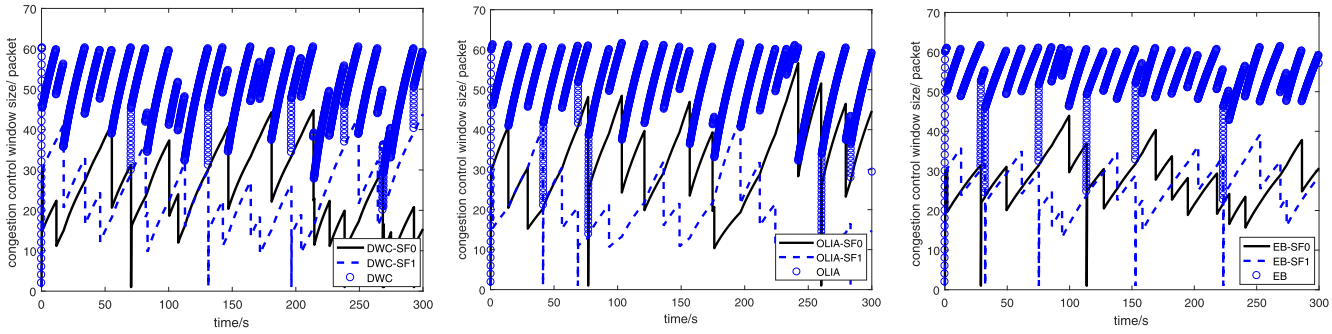
(c) Scenario 3:SF2 and SF3 share the bottleneck 2, while SF1 traverses distinct bottleneck



(d) Scenario 4: The bottleneck shift from bottleneck 1 to bottleneck 2

Fig. 6. Basic topologies for SB-CC evaluation.

2) Fairness Analysis: In Scenario 1, as shown in Fig. 6(a), SF_1 and SF_2 share the same bottleneck with the TCP flow. We add background traffic through Router 1 to construct the situation that Router1 is a congested bottleneck point. In order to achieve fairness with TCP, both bottleneck fairness and network fairness principles will limit MPTCP's aggregated throughput similar to a single-path TCP through the same bottleneck. Thus, these MPTCP algorithms should make the



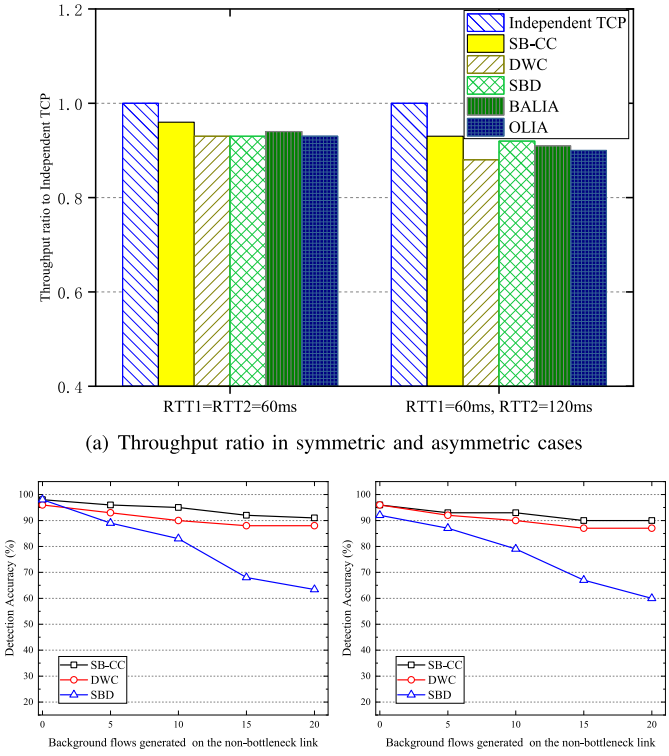
(a) CWND change for DWC (b) CWND change for SBD (c) CWND change for SB-CC

Fig. 7. Congestion window fluctuation of DWC, SBD and SB-CC.

aggregated throughput of all the MPTCP subflows equal to that with TCP. In Scenario 1 of Fig. 6 (Fig. 6(a)), there is a single shared bottleneck between the two MPTCP subflows (SF_1 and SF_2) and a single-path TCP flow is also involved through the shared bottleneck. We conduct the throughput evaluation of SB-CC, DWC, SBD, BALIA, and OLIA, and set TCP's average throughput as measurement baseline. In this scenario, we conduct the tests in two cases: symmetric case and asymmetric case.

Fig. 8 shows the ratios of T_{TCP}, T_{SB-CC}, T_{DWC}, T_{SBD}, T_{BALIA}, and T_{OLIA} to T_{TCP}, *i.e.*, 1, T_{SB-CC}/T_{TCP}, T_{DWC}/T_{TCP}, T_{SBD}/T_{TCP}, T_{BALIA}/T_{TCP}, and T_{OLIA}/T_{TCP}. We can see that no matter whether the RTTs in different paths are symmetric or asymmetric, all these algorithms can guarantee that the MPTCP connection gain nearly the same throughput as the TCP flow. Therefore, bottleneck fairness can be achieved, as all MPTCP subflows and TCP flows go through the same bottleneck, and in this scenario, bottleneck fairness is consistent with network fairness. Moreover, as shown in Fig. 8(a), both T_{DWC} and T_{SB-CC} are close to T_{TCP}, which indicates that SBD and SB-CC can detect shared bottleneck more accurately than other compared schemes, and achieve a better bottleneck fairness. Then, we further increase background flows of the links other than the shared bottleneck and observe its impact on the shared bottleneck detection accuracy. Fig. 8(b) shows the detection accuracy respectively with SB-CC, DWC, and SBD in the symmetric scenario. As shown in this figure, different background flows of the links other than the shared bottleneck, in all aspects, affect the OWD measurement result and its variation trend, which easily results in shared bottleneck detection errors in SBD. As shown in Fig. 8(c), in the asymmetric scenario, we can draw a similar conclusion as that in Fig. 8(b).

3) Throughput Analysis: In Scenario 2 of Fig. 6 (Fig. 6(b)), two disjoint paths have different bottlenecks and there is no shared bottleneck between the two subflows in the MPTCP connection. Two single-path TCP connections are introduced to compete with MPTCP in each path. In this scenario, each MPTCP subflow should behave like a single-path TCP to achieve bottleneck fairness, and the aggregated throughput of SB-CC, SBD, and DWC should be similar to $T_{TCP_1} + T_{TCP_2}$ theoretically. Meanwhile, with a network fairness based



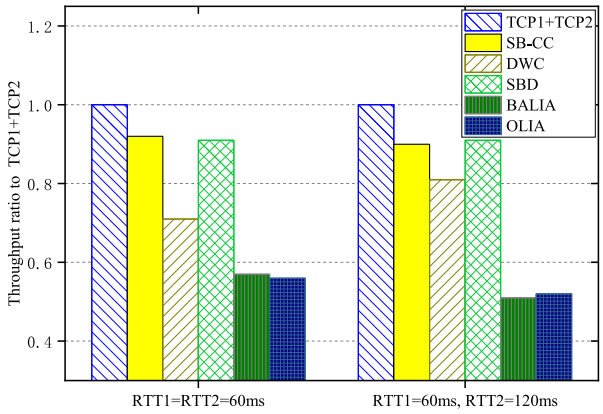
(a) Throughput ratio in symmetric and asymmetric cases

(b) Detection accuracy in symmetric case (c) Detection accuracy in asymmetric case

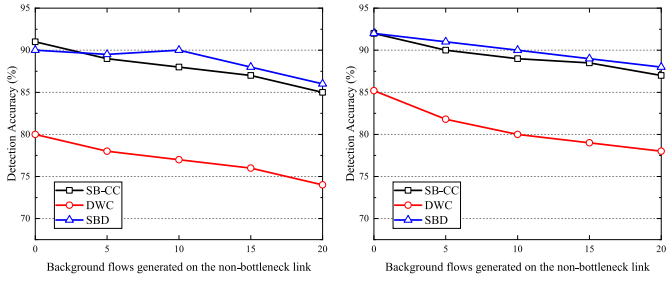
Fig. 8. Throughput ratio and detection accuracy for Scenario1.

coupled congestion control algorithm, like BALIA and OLIA, the aggregated throughput should be close to the best one of TCP_1 and TCP_2.

As shown in Fig. 9(a), the aggregated throughput of BALIA and OLIA are nearly 50% of that of $TCP_1 + TCP_2$, while SB-CC, DWC, and SBD are much better than BALIA and OLIA in terms of throughput. Especially, the throughput of SB-CC and SBD are close to that of $TCP_1 + TCP_2$. It means that SB-CC and SBD achieve bottleneck fairness through correctly freeing SF_1 and SF_2 to behave like two single-path TCP connections. Furthermore, from Fig. 9(a), we can see that SB-CC and SBD always outperform DWC no matter whether it's in a symmetric scenario or in an asymmetric scenario. Especially, in the symmetric scenario with similar background



(a) Throughput ratio in symmetric and asymmetric cases



(b) Detection accuracy in symmetric case

(c) Detection accuracy in asymmetric case

Fig. 9. Throughput ratio and detection accuracy for Scenario2.

traffic overhead, the advantages of SB-CC and SBD are more obvious. Then, we further increase the background flows of the links other than the two bottlenecks and observe its impact on the shared bottleneck detection accuracy. Fig. 9(b) shows the detection accuracy of SB-CC, DWC, and SBD in the symmetric scenario. For DWC, in an observation window, subflows with packet loss or increased delay may not pass through a shared bottleneck. Therefore, as shown in Fig. 9(b), DWC's shared bottleneck detection accuracy is much lower than the two other schemes in this scenario. Our solution is based on ECN mechanism, which can accurately represent the bottleneck queue length, so it is less affected by background flows, and the detection accuracy is improved. In the case of background flows with light traffic, the performance of our proposed SB-CC is very good and SBD is equally good. As the traffic of background flows increases, the detection errors in SB-CC and SBD also increase, but the detection accuracy of these two schemes is still much higher than that in DWC. As shown in Fig. 9(c), in the asymmetric scenario, we can also draw a similar conclusion as that in Fig. 9(b).

Fig. 10 shows the throughput performance with respect to the increase of packet loss rate on both subflows in the symmetric scenario where RTT on both paths is 60ms. As shown in Fig. 10, the throughputs of SB-CC, DWC, and SBD all are close to that of $TCP_1 + TCP_2$ when the loss rate is very small. However, the throughput of DWC reduces to nearly 50% of that of $TCP_1 + TCP_2$ when the loss rate increases to 4%, while the throughputs of SB-CC and SBD still remain nearly equal to that of $TCP_1 + TCP_2$. We observe

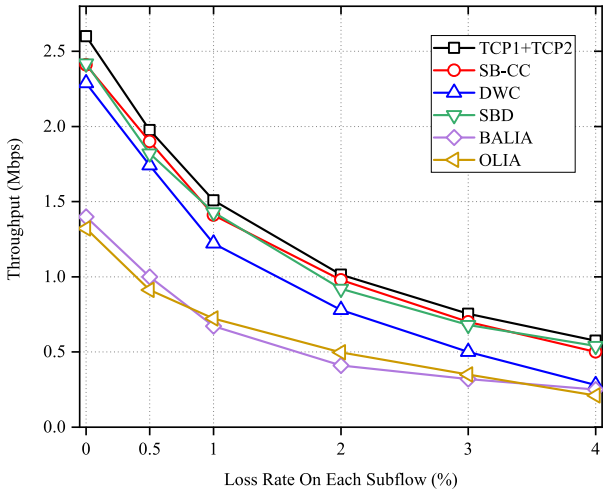


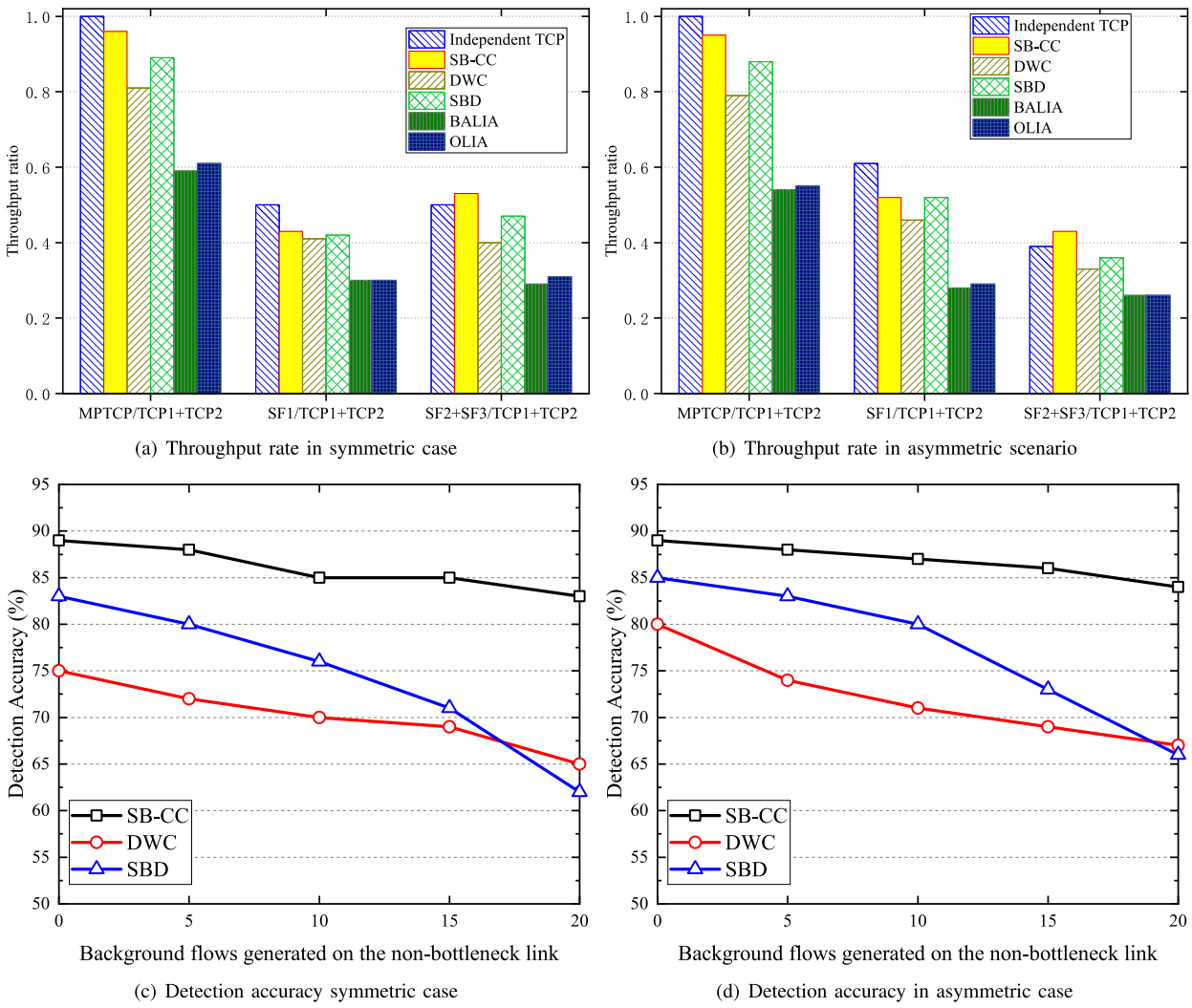
Fig. 10. Comparison of average throughput with varying loss rate.

that SB-CC and SBD obviously outperform DWC and the gap becomes larger as the loss rate increases. In the lossy network, for DWC, random packet loss will increase the probability of mis-detection, and moreover, the higher the packet loss rate is, the higher the probability of mis-detection will be. The detection accuracy in SB-CC is not affected by the presence of packet loss rate, as ECN marking is conducted based on the observation of the queue length at intermediate routers, which is not affected by the random packet loss.

The experiments in Scenario 2 shows that T_{SB-CC} is closer to $T_{TCP_1} + T_{TCP_2}$, which means that each of the disjoint subflows can reach the throughput of single-path TCP and a good bottleneck fairness can be achieved.

4) Performance Analysis in Mixed Scenario: Scenario 3 in Fig. 6 (Fig. 6(c)) is the mixture of Scenario 1 and Scenario 2. Besides, we also add a random packet loss rate of 0.5% in this scenario. SF2 and SF3 share a bottleneck with TCP_2, and meanwhile SF_1 runs independently. Based on the bottleneck fairness principle, SF_2 and SF_3 should be grouped together and achieve equal aggregate throughput as a regular TCP, while SF_1 could get throughput similar to that of a single path TCP. Thus, bottleneck-fairness based coupled congestion control algorithms in this scenario should theoretically achieve aggregated throughput equal to that of $TCP_1 + TCP_2$.

Fig. 11(a) and Fig. 11(b) show the throughput performance of SB-CC, DWC, SBD, BALIA, and OLIA with respect to the ratios of T_{MPTCP} (denoted as the total throughput over SF1, SF2, and SF3), T_{SF1}, and $T_{SF2+SF3}$ to $T_{TCP1+TCP2}$, respectively. We can see that SBD and our proposed SB-CC perform almost as well as the combination of TCP_1 and TCP_2, achieving better throughput performance than BALIA which is limited by network fairness. This is because SB-CC and SBD can correctly distinguish the bottleneck set and disjoint subflows, thereby further providing the corresponding congestion control toward these subflows. Furthermore, SB-CC outperforms SBD in terms of throughput as SB-CC can timely adjust the congestion window according to the actual congestion degree by using ECN mechanism. Meanwhile, in SBD, the random packet loss will cause the



(a) Throughput rate in symmetric case

(b) Throughput rate in asymmetric scenario

(c) Detection accuracy symmetric case

(d) Detection accuracy in asymmetric case

Fig. 11. Throughput ratio and detection accuracy for Scenario3.

performance degradation. Fig. 11(c) and Fig. 11(d) further show the impact of the background traffic of the links other than the two bottlenecks and observe its impact on the shared bottleneck detection accuracy in Scenario 3. For DWC, in an observation window, subflows with packet loss or increased delay may not pass through the same bottleneck, but may pass through different bottlenecks. The presence of random packet loss further increase the probability of mis-detection. Therefore, as shown in Fig. 11(c) and Fig. 11(d), DWC's detection accuracy is much lower than that in other compared schemes in this scenario. Meanwhile, although we make SF_2 and SF_3 share the same bottleneck, different and continuously changed background flows on non-bottleneck links may also affect the variation tendency of the OWD measurment result, which will lead to shared bottleneck detection errors in SBD. SB-CC relies on the ECN mechanism to reflect the congestion level of the bottlenecks, which will not be affected by background traffic on non-bottleneck links, and the mis-detection probability can be further reduced by the two-stage detection.

5) Shifting Bottleneck Analysis: Moreover, we further conduct the test for Scenario 4 in Fig. 6 (Fig. 6(d)) to verify the effectiveness of our proposed SB-CC in shared bottleneck detection. In this scenario, we design a case with bottleneck shifting. Before bottleneck shifting at t = 30s, background traffic traverses Router1, and SF_1 and SF_2 share a bottleneck with TCP1. At t = 30s, we move background traffic from traversing Router 1 to traversing Router 5, and then SF_2 and SF_3 share a bottleneck with TCP_2. For this scenario, Fig. 12 shows the shared bottleneck detection accuracy in SB-CC and SBD. As SBD's detection cycle is longer, when bottleneck shifting happens, we can observe in Fig. 12 that SBD achieves very low detection accuracy at the beginning, and needs about 30s to get close to that of our proposed SB-CC. Therefore, SB-CC has stronger timeliness than SBD, and in the current Internet scenarios with time-varying traffic, this feature is important.

B. Performance Analysis of SB-FPS

In this section, we evaluate our proposed scheduling mechanism on NS3 simulator [38]. The MPTCP NS3 code is provided by google MPTCP group [39]. Another two scheduling mechanisms, LowRTT and FPS are implemented as comparisons. The main difference between FPS and SB-FPS is

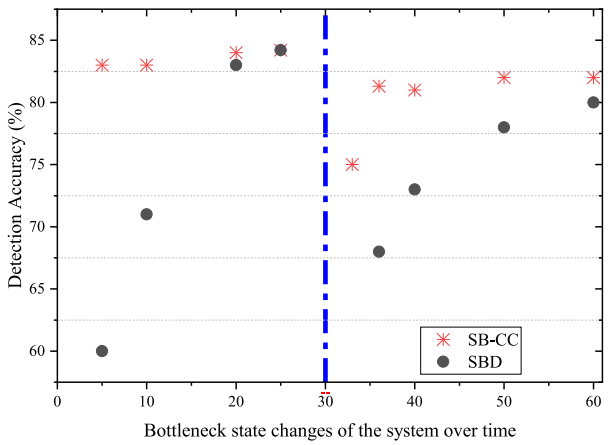


Fig. 12. Detection accuracy of SBD and SB-CC before and after the bottleneck shift at t = 30s.

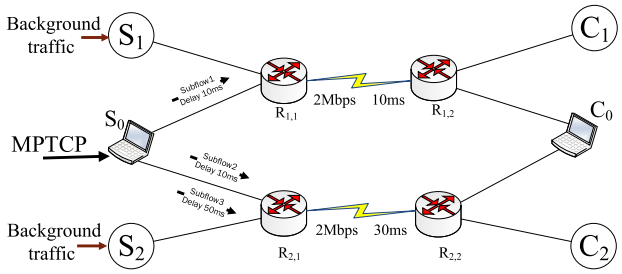


Fig. 13. Simulation setup.

the algorithm to estimate scheduling value. Besides, we use SB-CC as the congestion control mechanism since we only want to compare the performance of each scheduling mechanism in this section.

1) Simulation Setup: The simulation scenario is shown in Fig. 13. Three subflows are established between the MPTCP client (C_0) and the server (S_0) in such a way that SF_2 and SF_3 share a bottleneck, while SF_1 runs independently. SF_1 has 2Mbps bandwidth and 20ms latency, while the delay of SF_2 and SF_3 are 10ms and 50ms, respectively. A synthetic mix of TCP and UDP are generated between client (C_1, C_2) and server (S_1, S_2) as background traffic creating a more realistic emulation environment. The background traffic compete with MPTCP flow at bottleneck. SF_2 and SF_3 share the bottleneck with 2Mbps bandwidth and 30ms latency. The total delay of the two subflows are 40ms and 80ms, respectively.

2) SB-FPS Performance Analysis: Fig. 14 shows the throughput performance with respect to the transmitted file size changing by using different schedulers in MPTCP in the given asymmetric scenario. As shown in Fig. 14, LowRTT performs the worst regardless of the data volume. The reason is that all packets transmitted through the fast path have to wait for the packets transmitted from the slow path. Then, the throughput of the subflow on the fast path is dragged down to the same level as that of the subflow on the slow path. FPS schedules packets according to the ratio of estimated RTTs and the current window size such that it can make more use of the fast path, so FPS consistently outperforms LowRTT in terms of throughput. As SB-FPS provides a more fine-grained

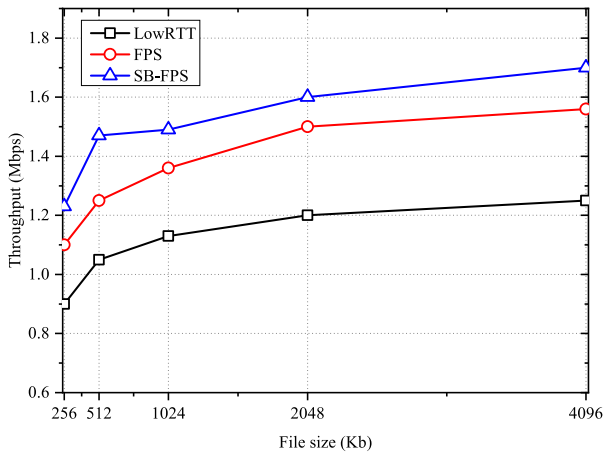


Fig. 14. The contrast of throughput with the file size change.

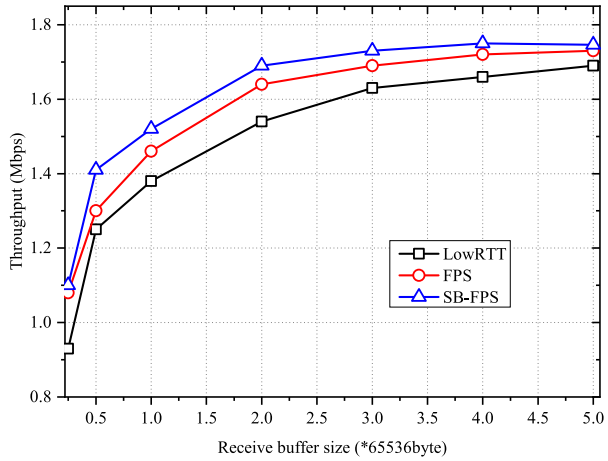


Fig. 15. The contrast global throughput with the packet receiving buffer size change.

scheduling mechanism, which can estimate the scheduling value N more accurately, SB-FPS further performs much better than FPS. From the results of these two experiments, we observe that

- SB-FPS has the highest transmission throughput, compared to other scheduling algorithms.
- Throughput is significantly affected by out-of-order receptions. We can expect an even lower throughput with a smaller size buffer.

Fig. 15 shows the throughput performance with respect to the receive buffer size changing by using different schedulers in MPTCP in the given asymmetric scenario. When the receive buffer size is small, MPTCP's throughput will be severely affected. With the increase of the receive buffer size, the throughput of MPTCP grows quickly in the very beginning. As the receive buffer size continues to increase, the increase in throughput will be slower and slower. And when the receive buffer is large enough to accommodate all the received packets, the throughput will not increase any more since buffer-blocking-caused retransmissions will be greatly reduced.

Furthermore, Fig. 16 shows the number of out-of-order packets varying with the receive buffer size changing by

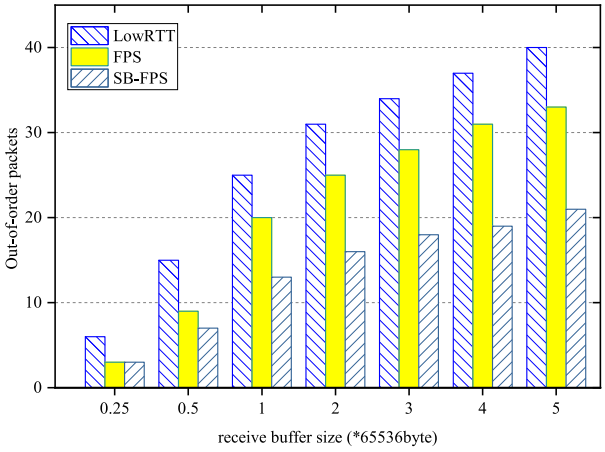


Fig. 16. The contrast of out of ordered packets with respect to packet receiving buffer size.

using different schedulers in MPTCP in the given asymmetric scenario. From the figure, we can see that the number of out-of-order packets when using SB-FPS is much less than that when using LowRTT or FPS regardless of the receive buffer size. Furthermore, SB-FPS outperforms FPS as it schedules packets according to the state of each subflow and estimates the scheduling value in more fine-grained control.

V. Conclusion

In this paper, we consider both congestion control and scheduling algorithm to further improve the performance of MPTCP. We first proposed SB-CC, a new *Bottleneck Fairness* based congestion control mechanism for MPTCP. It provides an explicit bottleneck detection mechanism by using ECN. Meanwhile, with the subflow congestion degree, SB-CC dynamically adjusts the congestion window to make better use of bottleneck resources and smooth out the window fluctuation. We further put forward a fine-grained Forward Prediction packet Scheduling scheme based on Shared Bottleneck detection, called SB-FPS. Our experimental results prove that the SB-CC provides higher throughput performance than existing MPTCP congestion control algorithms while still maintains fairness with TCP at shared bottleneck. Besides, simulation results verify that SB-FPS plays an important role in mitigating the head-of-line blocking problem.

Acknowledgment

The authors sincerely thank the editor Dr. A. Kuzmanovic and the anonymous referees for their invaluable suggestions that have led to the present improved version from the original manuscript.

References

[1] D. Ma and M. Ma, "A QoS oriented vertical handoff scheme for WiMAX/WLAN overlay networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 4, pp. 598–606, Apr. 2012.

[2] S. Prabhavat, H. Nishiyama, N. Ansari, and N. Kato, "Effective delay-controlled load distribution over multipath networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 10, pp. 1730–1741, Oct. 2011.

[3] M. Allman, V. Paxson, and E. Blanton, *TCP Congestion Control*, document RFC 5681, IETF, 2009.

[4] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, *TCP Extensions For Multipath Operation With Multiple Addresses*, document RFC 6824, IETF, 2013.

[5] C. Raiciu, D. Wischik, and M. Handley, "Practical congestion control for multipath transport protocols," Univ. College London, London, U.K., Tech. Rep, 2009. Accessed: Jan. 19, 2020. [Online]. Available: http://nrg.cs.ucl.ac.uk/mptcp/mptcp-techreport.pdf

[6] C. Raiciu, M. Handley, and D. Wischik, *Coupled Congestion Control for Multipath Transport Protocols*, document RFC 6356, IETF, 2011.

[7] 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.

[8] Y. Cao, M. Xu, and X. Fu, "Delay-based congestion control for multipath TCP," in *Proc. 20th IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2012, pp. 1–10.

[9] 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.

[10] K. Xue, J. Han, H. Zhang, K. Chen, and P. Hong, "Migrating unfairness among subflows in MPTCP with network coding for wired-wireless networks," *IEEE Trans. Veh. Technol.*, vol. 66, no. 1, pp. 798–809, Mar. 2016.

[11] S. Hassayoun, J. Iyengar, and D. Ros, "Dynamic window coupling for multipath congestion control," in *Proc. 19th IEEE Int. Conf. Netw. Protocols*, Oct. 2011, pp. 341–352.

[12] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 63–74, Aug. 2010.

[13] V. Jeyakumar, M. Alizadeh, and Y. E. A. Geng, "Millions of little minions: Using packets for low latency network programming and visibility," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 3–14, 2014.

[14] S. Floyd, D. K. K. Ramakrishnan, and D. L. Black, *The Addition of Explicit Congestion notification (ECN) to IP*, document RFC 3168, IETF, 2013.

[15] D. Murray, T. Koziniec, S. Zander, M. Dixon, and P. Koutsakis, "An analysis of changing enterprise network traffic characteristics," in *Proc. 23rd Asia–Pacific Conf. Commun. (APCC)*, Dec. 2017, pp. 1–6.

[16] *QoS: Congestion Avoidance Configuration Guide, Cisco IOS XE Release 3S*. Accessed: Jan. 19, 2020. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/iosxml/ios/qosconavd/configuration/xe-3s/qos-conavd-xe-3s-book/qosconavd-wred-ecn.html

[17] J. Iyengar, P. Amer, and R. Stewart, "Receive buffer blocking in concurrent multipath transfer," in *Proc. IEEE Global Telecommun. Conf.*, Nov./Dec. 2005, pp. 1–6.

[18] 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.

[19] S. Ferlin, O. Alay, O. Mehani, and R. Boreli, "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks," in *Proc. IFIP Netw. Conf. Workshops (IFIP)*, May 2016, pp. 431–439.

[20] 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.

[21] 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.

[22] M. Handley, J. Padhye, and S. Floyd, *TCP Congestion Window Validation*, document RFC 2861, IETF, 2000.

[23] M. Honda, Y. Nishida, L. Eggert, P. Sarolahti, and H. Tokuda, "Multipath congestion control for shared bottleneck," in *Proc. 7th Int. Workshop Protocols Future, Large-Scale Diverse Netw. Transp.*, May 2009, pp. 19–24.

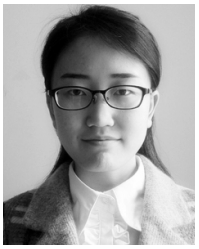
[24] W. Wei, Y. Wang, K. Xue, D. S. L. Wei, J. Han, and P. Hong, "Shared bottleneck detection based on congestion interval variance measurement," *IEEE Commun. Lett.*, vol. 22, no. 12, pp. 2467–2470, Dec. 2018.

[25] 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.*, Apr. 2016, pp. 1–9.

[26] M. Zhang, J. Lai, A. Krishnamurthy, L. L. Peterson, and R. Y. Wang, "A transport layer approach for improving end-to-end performance and robustness using redundant paths," in *Proc. USENIX Annu. Tech. Conf. (ATC)*, 2004, pp. 99–112.
[27] M. S. Kim, T. Kim, Y. Shin, S. S. Lam, and E. J. Powers, "A wavelet-based approach to detect shared congestion," *IEEE/ACM Trans. Netw.*, vol. 16, no. 4, pp. 763–776, Mar. 2008.
[28] M. M. Yousaf and M. Welzl, "On the accurate identification of network paths having a common bottleneck," *Sci. World J.*, vol. 2013, Nov. 2013, Art. no. 890578.
[29] C. Xu, Z. Li, J. Li, H. Zhang, and G.-M. Muntean, "Cross-layer fairness-driven concurrent multipath video delivery over heterogeneous wireless networks," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 25, no. 7, pp. 1175–1189, Jul. 2015.
[30] S. Barré *et al.*, "Implementation and assessment of modern host-based multipath solutions," Ph.D. dissertation, Dept. Comput. Sci. Eng., Catholic Univ. Louvain, Louvain-la-Neuve, Belgium, 2011.
[31] *Linux MPTCP Kernel*. Accessed: Jan. 19, 2020. [Online]. Available: http://www.multipath-tcp.org/
[32] 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.*, Sep. 2010, pp. 1–5.
[33] S. Floyd. *RED: Discussions of Setting Parameters*. Accessed: Jan. 19, 2020. http://www.icir.org/floyd/REDparameters.txt
[34] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. 8th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2011, pp. 99–122.
[35] H. Han, S. Shakkottai, C. V. Hollot, R. Srikant, and D. Towsley, "Multi-path TCP: A joint congestion control and routing scheme to exploit path diversity in the Internet," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1260–1271, Dec. 2006.
[36] Q. Liu, K. Xu, H. Wang, and L. Xu, "Modeling multi-path TCP throughput with coupled congestion control and flow control," in *Proc. 18th ACM Int. Conf. Model., Anal. Simulation Wireless Mobile Syst. (MSWiM)*, 2015, pp. 99–106.
[37] A. Botta, A. Dainotti, and A. Pescapé, "A tool for the generation of realistic network workload for emerging networking scenarios," *Comput. Netw.*, vol. 56, no. 15, pp. 3531–3547, Oct. 2012.
[38] *NS3 Simulator*. Accessed: Jan. 19, 2020. [Online]. Available: https://www.nsnam.org/
[39] *MPTCP NS3 Code*. Accessed: Jan. 19, 2020. [Online]. Available: http://code.google.com/p/mptcp-ns3/

Wenjia Wei received the B.S. degree from the School of Information Science and Engineering in 2013. He is currently pursuing the Ph.D. degree in information and communication engineering with the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC). His research interests include future Internet architecture design and transmission optimization.

Kaiping Xue (Senior Member, IEEE) received the bachelor'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 May 2012 to May 2013, he was a Post-Doctoral Researcher with the Department of Electrical and Computer Engineering, University of Florida. He is currently an Associate Professor with the School of Cyber Security and the Department of EEIS, USTC. He has authored or coauthored more than 80 technical articles in the areas of communication networks and network security. His research interests include next-generation Internet, distributed networks, and network security. He is a Fellow of IET. His work won best paper awards in the IEEE MSN 2017 and the IEEE HotICN 2019 and the Best Paper Runner-Up Award in the IEEE MASS 2018. He is also serving as the Program Co-Chair for the IEEE IWCMC 2020 and SIGSAC@TURC 2020. He serves on the Editorial Board for several journals, including the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS (TWC), the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT (TNSM), *Ad Hoc Networks*, IEEE ACCESS, and *China Communications*. He has also served as a Guest Editor for the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (JSAC) and the Lead Guest Editor for the *IEEE Communications Magazine*.



Jiangping Han received the B.S. degree from the Department of Electronic Engineering and Information Science (EEIS), USTC, in July 2016, where she is currently pursuing the Ph.D. degree in communication and information systems. Her research interests include future Internet architecture design and transmission optimization.

David S. L. Wei (Senior Member, IEEE) received the Ph.D. degree in computer and information science from the University of Pennsylvania in 1991. From May 1993 to August 1997, he was on the Faculty of Computer Science and Engineering, The University of Aizu, Japan, as an Associate Professor and then a Professor. He is currently a Professor of the Computer and Information Science Department, Fordham University. He has authored or coauthored more than 120 technical articles in various archival journals and conference proceedings. His research interests include cloud computing, big data, the IoT, and cognitive radio networks. He was a Guest Editor or the Lead Guest Editor for several special issues in the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, the IEEE TRANSACTIONS ON CLOUD COMPUTING, and the IEEE TRANSACTIONS ON BIG DATA. He also served as an Associate Editor for the IEEE TRANSACTIONS ON CLOUD COMPUTING from 2014 to 2018 and the *Journal of Circuits, Systems and Computers* from 2013 to 2018.

Peilin Hong received the B.S. and M.S. degrees from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 1983 and 1986, respectively. She is currently a Professor and an Advisor for Ph.D. candidates with the Department of EEIS, USTC. She has published two books and over 150 academic articles in several journals and conference proceedings. Her research interests include next-generation Internet, policy control, IP QoS, and information security.