TCCC: A Throughput Consistency Congestion Control Algorithm for MPTCP in Mixed Transmission of Long and Short Flows

Jiayu Yang, *Graduate Student Member, IEEE*, Jiangping Han, *Member, IEEE*, Kaiping Xue, *Senior Member, IEEE*, Yansen Wang, Jian Li, *Member, IEEE*, Yitao Xing, *Graduate Student Member, IEEE*, Hao Yue, *Member, IEEE*, and David S. L. Wei, *Life Senior Member, IEEE*

***Abstract*—Existing congestion control algorithms for MPTCP that care about only long flow transmission aim at the Congestion-Avoidance (CA) phase and they need a long time to reach convergence states. We verified that the exponential growth of congestion window (cwnd) in the uncoupled Slow-Start (SS) leads to not only unfairness to TCP but also buffer overflow due to burst data. Moreover, these algorithms cannot support fair bandwidth sharing among TCP/MPTCP flows before reaching convergence at the bottleneck, which may reduce the transmission efficiency of short flows and even hurts long flows. In this paper, we propose a Throughput Consistency Congestion Control (TCCC) algorithm consisting of Coupled Slow-Start (CSS) and Aggressive Congestion Avoidance (ACA). To prevent packet loss caused by excessive burst data, CSS couples the increment of subflows' cwnd and reset the ssthresh value to safely move the flows to CA when it achieves expected throughput. Based on CSS, ACA periodically detects path states and allocates the same throughput increment as the best TCP to subflows to achieve fair bandwidth share in CA. Finally, we implement TCCC in both NS3 and real testbed. The results show that TCCC reduces retransmissions, improves transmission efficiency, and maintains better fairness.**

***Index Terms*—Multi-path TCP, congestion control, transmission efficiency, fairness.**

I. Introduction

With the rapid development of various network access technologies, network access devices are usually equipped with multiple interfaces. For instance, mobile devices have WiFi and 3G/4G wireless interfaces, and data center networks often provide alternative paths between two hosts [1]. Though transmission efficiency can be improved when multiple interfaces are utilized simultaneously, TCP can only transmit over one interface at a time. To provide the capability of simultaneous use of multiple paths between multi-homed end-hosts, Multi-path TCP (MPTCP) is recommended by IETF [2]. It is not only compatible with regular TCP but also able to aggregate bandwidth and improve robustness.

Congestion Control is one of the key issues affecting the transmission efficiency of MPTCP. It controls the data injected into the network through the congestion window (cwnd) to reduce packet loss and retransmissions caused by congestion. Usually, congestion control algorithms consist of two parts including Slow-Start (SS) and Congestion-Avoidance (CA) with disparate cwnd adjustment strategies. Most algorithms grow cwnd exponentially to rapidly increase the transmission rate in SS. In CA, to prevent congestion, they augment cwnd with a slower pace. Raiciu et al. [3] proposed that MPTCP congestion control algorithms should satisfy the following design goals: (1) Improve throughput: MPTCP should achieve at least the same throughput as a single TCP connection when there is no congestion. (2) Do no harm: MPTCP takes up no more capability than regular TCP at the shared bottleneck.

So far, many popular congestion control algorithms of MPTCP have been proposed to achieve the two goals [3], [4], [5]. However, these algorithms only couple cwnd growth of subflows in the CA phase, and each subflow still behaves like independent TCP in SS. Thus, compared with TCP that may cause buffer overflow at routers due to the exponential increase of cwnd in SS, MPTCP suffers more severe packet loss (with multiple subflows in SS). This is adverse to the transmission efficiency of short flows, which become more and more popular in the current network environment [6] and account for 95% of real Multi-path TCP traffic [7]. Moreover, although many algorithms aim to satisfy the second design goal, the performance of MPTCP is restricted. Their throughput is much less than that of a TCP flow before reaching convergence when they compete at the same bottleneck [8]. Therefore, these algorithms, with uncoupled SS and excessive restrictions on cwnd in CA, will undoubtedly

Manuscript received 14 March 2022; revised 24 July 2022; accepted 22 January 2023. Date of publication 6 February 2023; date of current version 9 October 2023. The work is supported in part by the National Natural Science Foundation of China (NSFC) under Grant No. 61972371, and Youth Innovation Promotion Association of the Chinese Academy of Sciences (CAS) under Grant No. Y202093. The associate editor coordinating the review of this article and approving it for publication was C. Avin. *(Corresponding authors: Jiangping Han; Kaiping Xue.)*

Jiayu Yang, Jiangping Han, Kaiping Xue, Jian Li, and Yitao Xing are with the School of Cyber Science and Technology, University of Science and Technology of China, Hefei 230027, Anhui, China (e-mail: jphan@ustc.edu.cn; kpxue@ustc.edu.cn).

Yansen Wang is with the Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei 230027, Anhui, China, and also with the Natural Language Processing Research Department, Meituan Group, Beijing 100102, China.

Hao Yue is with the Department of Computer Science, San Francisco State University, San Francisco, CA 94132 USA.

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

Digital Object Identifier 10.1109/TNSM.2023.3242419

reduce the transmission efficiency of short flows and affect the performance of long flows.

Although some works try to solve the above problems, they mainly focus on one of the scenarios of long flows or short flows. For example, to improve the performance of short flows using MPTCP, Barik et al. [9] and Li et al. [10] proposed different methods to slow down the cwnd increment of MPTCP in SS. There are also schemes dedicated to improving the transmission efficiency of both long and short flows. Thomas et al. [11] speeds up the convergence process of MPTCP algorithms, which couples cwnd increment in both SS and CA phases. Kheirkhah et al. [12] and Lee et al. [13] only work in special environments of data centers and 5G networks. Additionally, some algorithms use emerging machine learning technology [14], [15], [16], [17]. Compared with heuristic algorithms, they may adjust cwnd more accurately, however, having high overhead and facing the problem of controlling lags.

The real network, with the requirement to transmit data of various sizes, is usually a mixed transmission of long flows and short flows with disparate characteristics. The existing way of judging short flow and long flow is empirical. Specifically, some related works distinguish them by the file size, for example, Zhao et al. [18] and Kheirkhah et al. [19] illustrated that a majority (90%) of the flows are short flows with a size smaller than 1MB. Some suggest that short flows can be judged by completion time, which may never leave the Slow-Start phase with a small congestion window [20]. In our scheme, short flows are connections transmitting files smaller than 1MB, whose transmission efficiency is mainly affected by the SS algorithm. Long flows, with larger transmission files, consume most of their life during the CA phase and are mainly affected by the CA algorithm. Besides, fairness (bottleneck fairness) is also one of the important metrics in our design, which aims to enable MPTCP flows to achieve the same throughput as TCP flows at the bottleneck [21].

In this paper, we jointly consider fairness transmission for both short and long flows and propose a Throughput Consistency Congestion Control (TCCC) scheme. It is a heuristic algorithm that consists of two parts: Coupled Slow-Start (CSS) and Aggressive Congestion-Avoidance (ACA) algorithms. Mainly, TCCC couples the increment of cwnd in both SS and CA phases based on the throughput consistency principle, which means, TCCC is committed to ensuring that MPTCP achieves the same throughput as the best TCP. Specifically, when multiple subflows are in the SS phase, CSS couples the cwnd growth of each subflow to prevent buffer overflow caused by high-speed window increase. Then, all the subflows actively move to the CA phase after reaching the maximum throughput limitation to ensure that MPTCP achieves the same throughput as TCP in the whole SS process. Based on CSS, the ACA algorithm updates the coupling coefficient for window growth periodically. It records the path state of each subflow within an updating interval and computes the estimated throughput increment of MPTCP according to the best TCP flow. Then, it allocates throughput increments to subflows and updates their coupling coefficients. The main contributions of this paper are summarized as follows:

- To improve the performance of MPTCP in the mixed transmission of long and short flows while ensuring fairness with TCPs, we propose a TCCC scheme composed of CSS and ACA. CSS couples the exponential growth factor of multiple subflows to avoid severe packet losses in SS and improves the efficiency of short flows. ACA periodically maintains the throughput increment of all subflows in MPTCP equal to the best TCP flow's to achieve fairness. It provides new adjustment factors of cwnd to strive for high throughput and better performance for long flows.
- We further analyze the fairness of the TCCC by using a fluid model first proposed by Peng et al. [5]. By comparing the aggressiveness of each algorithm through the provided utility coefficients, we reveal that TCCC can maintain fairness with TCP.
- We implement TCCC in both software simulation and real testbed and evaluate the performance of short and long flows. The results reveal that TCCC can bring a goodput gain of up to 40% for short flows, and increases the bandwidth share of MPTCP at the bottleneck to 97.9% of TCP for long flows.

This paper inherits the basic idea of our conference paper [22]. They differ mainly in the following aspects: We further consider the scenario of long flow transmission, which is out of the scope of CSS but is important for the current Internet, and provide a new algorithm called ACA. Thus, TCCC can improve the efficiency of MPTCP in mixed long and short flows, and maintain fairness with TCP flows at the same bottleneck. We also provide the theoretical analysis of fairness performance for CSS and ACA and reproduce the original experiments. Meanwhile, we expand the experimental testing from software simulation to the real testbed and reveal the efficiency of our proposed algorithm through various tests.

The rest of this paper is organized as follows. In Section II, we introduce several popular congestion algorithms and illustrate the performance issues of the current MPTCP. Our proposed TCCC is described in detail in Section III and we present the theoretical analysis in Section IV. Subsequently, we give a performance evaluation in Section V and conclude our work in Section VI.

II. Related Work and Problem Statement

Congestion control is the key issue in transport protocol design, which plays a crucial role in efficient transmissions. TCP congestion control has developed rapidly from TCP Reno [23], NewReno [24] to S-TCP [25], CUBIC [26] and till now [27], [28]. These algorithms treating packet loss as the notification of path congestion are classified as loss-based ones, which decrease window size if a loss occurs.

The loss-based congestion control algorithms of MPTCP are mostly extended from the existing TCP Reno algorithm. For example, SCTP [29] is a straightforward extension of the single-path TCP, which uses independent TCP Reno in each subflow and occupies multiple sources compared with TCP. Although simple, it will seriously occupy the bandwidth of the TCP flows at the same bottleneck. To maintain fairness with TCP flows, Equal-Cost Multi-Path TCP (EMTCP) [30] adds $\frac{1}{\sqrt{n}}$ (n is the number of subflows of MPTCP) to the adjustment coefficient of cwnd to increase the window more

gently. However, it does not consider differences between paths and would not make efficient use of the network. So another two methods Coupled [31] and LIA [3] are proposed, which remove traffic to the least congested path while also maintaining some in the congested one to improve transmission efficiency. Khalili et al. also proposed a new method OLIA [4], which adds a correction factor to the adjustment factor of LIA to achieve Pareto-optimal. Peng et al. [5] further demonstrated that there is a tradeoff between responsiveness and TCP-friendliness in MPTCP. So they put forward a more balanced solution called BALIA to ensure fairness with TCP.

Most of the congestion control algorithms mentioned above are optional algorithms applied in the MPTCP kernel and capable to maintain fairness between TCP and MPTCP to some extent. However, all of them only couple the cwnd increment factor of each subflow in the CA phase and behave like independent TCP in SS. Zhang et al. [32] revealed that a single TCP may cause buffer overflow at routers due to the exponential increase of cwnd in SS. It can be reasonably inferred that MPTCP with multiple uncoupled subflows in the SS phase is more likely to suffer severe packet loss. It becomes even inevitable as the increase of buffer size of commodity switches is significantly outpaced by that of the link speed [33]. The packets that are lost during transmission need to be retransmitted through dup-ACKs or waiting for Retransmission Time Out, which undoubtedly seriously affects the efficiency of short flows. Moreover, these algorithms also face problems at the CA phase as they excessively reduce the competitiveness of MPTCP to fully meet the second design goal of MPTCP. Kato et al. [8] conducted experiments in the Linux operating system and revealed that LIA is weaker than Reno [24]. Through experiments (illustrated in detail in Section V), we also perceive that LIA can only obtain about 80.7% of the theoretical bandwidth when it competes for bandwidth with TCP at the same bottleneck. BALIA and OLIA are also unable to achieve fair bandwidth allocation. Therefore, these algorithms that are unable to fairly share the bandwidth with TCP at the bottleneck also reduce the transmission efficiency of MPTCP for long flows.

So far, some work is devoted to solving the above problems, however, they are mainly focused on solving the problem of the SS or CA phase. For example, Barik et al. [9] noticed that uncoupled SS of MPTCP may lead to a buffer overflow, which is harmful to short flows. So they proposed a congestion control named LISA, which adjusts the initial value of cwnd according to the window size of existing subflows and keeps the sum of cwnd remains unchanged. In [10], Li et al. proposed Halfback, which suggests pacing out the data up to the threshold amount in one RTT in the JumpStart phase. They also provide reverse-ordered proactive retransmission notified by NACK to improve reactiveness if JumpStart fails. It needs routers to support the priority queue and NACK notification.

In [11], Thomas et al. discovered that most existing MPTCP algorithms suffer high convergence latency, rendering them ineffective for short flows. Thus, they proposed NMCC scheme, which achieves friendliness faster by normalizing the throughput growth rate of flows rather than the throughput itself. Moreover, they introduced the extended NMCC protocol that caters to friendliness upon both throughput growth and throughput reduction epochs.

In [34], Dong et al. proposed mVeno that modifies the addictive increment of Veno to effectively couple subflows. Specifically, mVeno calculates the backlog at the queue to notify network states and distinguishes packet losses caused by a random error of wireless links or by network congestion. Then, it provides different weights for subflows with disparate flow characteristics to improve the overall performance of MPTCP over wireless networks.

Wei et al. [35] utilized Explicit Congestion Notification (ECN) signals to detect congestion and discover the shared bottleneck flows. Then, they defined different congestion degrees in constrained additive increase and provided coupled congestion control algorithm with bottleneck-fairness. Moreover, they developed a forward prediction packet scheduling scheme to model the subflow's future behavior and preschedule data transmission to achieve fine-grained control.

MMPTCP [12] and DEFT [13] are two methods proposed to improve the transmission efficiency for both long and short flows for data centers and 5G networks, respectively. Though involving different traffic characteristics compared to an Internet-like network, their joint consideration of short and long flows in congestion control design is still worth introducing. MMPTCP runs in two phases. Initially, it scatters all the packets in the network under a single congestion window to exploit available paths. Short flows that ever are squeezed by the long flows can complete transmission faster. During the second phase, MMPTCP runs as standard MPTCP to pursue throughput. DEFT provides a novel window control algorithm considering the state of the millimeter, which shows fast responsiveness when the link changes. Besides, it includes a delay-equalizing algorithm that minimizes additional reordering delay in the receiver buffer.

There are also methods using emerging new techniques. Li et al. [14] and He et al. [17] both proposed a deep reinforcement learning framework to generate the congestion control rules for MPTCP. They conduct appropriate window adjustment factors in the online decision stage based on offline training. Xu et al. [15] provided a hierarchical tile coding algorithm to partition the high-dimensional state space into discrete tiles for state representation. Pokhrel and Walid [16] proposed a deep Q-learning-based congestion control scheme, which learns and acts from its experience of optimal congestion control using deep Q-learning. Despite more flexible window adjustment, these methods have a high training overhead and face the problem of controlling lag. Specifically, Xu et al. [15] and He et al. [17] both revealed that it cost an hour to several hours to train a model for a given network scenario. To improve transmission efficiency, they usually need to train model in multiple different network scenarios, which is a high overhead compared with traditional algorithms. When using the model, the online interference time varies from 0.5ms to 1ms, which causes controlling lags.

Compared with the algorithms mentioned above, the proposed TCCC focuses on improving the efficiency of

MPTCP on the premise of maintaining fairness with TCP connections. It simultaneously solves the problems of the two stages of MPTCP congestion control improving the transmission efficiency of short flows and long flows. However, if both types of flows are treated in the same way, long flows may block the transmission of short flows, which is a common problem for schemes that control sending rate at end nodes including ours. It can be solved by splitting flow queues at intermediate routers, which we leave as our future work.

III. Throughput Consistency Congestion Control

In this section, we introduce our Throughput Consistency Congestion Control (TCCC) scheme in detail. We elaborate on the two algorithms that work in the SS and CA phases, and explain the efficient improvement of the transmission for both long and short flows.

A. Design Overview

Firstly, we provide a design overview to present an overall idea of the scheme. TCCC consists of two parts including Coupled Slow-Start (CSS) and Aggressive Congestion-Avoidance (ACA) algorithms. They aim to solve the serious packet loss caused by the uncoupled window increase in SS, and the excessive restriction on cwnd growth in CA, respectively. Thus, the states of subflows determine which algorithm to use. Specifically, CSS works when MPTCP subflows are in the initial SS phase, which couples window increment and actively moves subflows to the CA phase, where ACA takes over control of window adjustment. Both CSS and ACA follow the principle of throughput consistency, that is, they strive to maintain the overall throughput of MPTCP consistent with that of the best TCP at the same bottleneck. Specifically, CSS couples the cwnd increment factor of multiple subflows in SS to avoid serious packet loss. When reaching the maximum throughput limitation, it actively moves to CA to maintain fairness in the SS process. Based on this, ACA periodically detects path states and maintains the expected throughput increment of MPTCP is equal to the best TCP flow within the updating interval. Then it allocates the throughput increment to subflows and provides new factors of cwnd adjustment to improve the aggressiveness of MPTCP when competing with TCP. We present the mathematical notation used in the system and derivation in **Table I**. The design details are shown in the following.

TABLE I
Mathematical Notation Used in System and Derivation

	Description
S	A set of all flows in a network
s	A flow in set S, $s \in S$
R_s	A set of all subflows of flow s
r	A subflow in set R_s, $r \in R_s$
L	A set of all available links in a network
l	A link in set L, $l \in L$
p_l	congestion price for link l
p_r	congestion price for route r
c_l	The finite capacity of link l.
y_l	The aggregate traffic rate of link l.
γ_l	A positive parameter of the fluid model determining the dynamic property.
H	A routing matrix describing the relationship of L and R
$rtt_{s,r}$	The round trip time of flow s on subflow r, for specific analysis of source s, written as rtt_r
$\min rtt$	The minimal rtt of flow s.
$cwnd_{s,r}$	Congestion window of flow s on subflow r, for specific analysis of source s, written as $cwnd_r$
$cwnd^{tcp}$	Congestion window of the TCP flow.
w_s	The vector $(cwnd_{s,r}, s \in S, r \in R_s)$
$Totcd$	The sum of congestion window size of flow.
$NewTotcd$	The sum of congestion window size of flow when a new subflow enters.
IW	The initial window size of a new subflow.
α	The coupling parameter of the SS phase.
CT	The estimated completion time of the SS phase.
$ssth$	The SS threshold of a TCP flow or an MPTCP subflow.
con_ssth	The SS threshold of an MPTCP flow.
cur_th	The throughput from current transmission data.
exp_th	The estimated throughput from ssthresh approximation.
MSS	The maximum segment size of TCP.
ini_SS	Flag that the flow is in initial SS.
n	The number of subflows in MPTCP connection.
i, j, k	The subscript of MPTCP subflow.
$x_{s,r}$	Transmission rate of flow s on subflow r,for specific analysis of source s, written as x_r
x_s	The vector $(x_{s,r}, s \in S, r \in R_s)$
T	The data updating period.
$I_r(w_s)$	The window increment of flow s on subflow r at the return of each ACK
$D_r(w_s)$	The window decrement of flow s on subflow r when a packet loss occurs
$k_r(x_s)$	Model parameter of flow s on subflow r determining the dynamic property of the algorithm.
$\phi_r(x_s)$	Model parameter of flow s on subflow r determining the equilibrium properties of the algorithm.
K_s	The vector $(k_r(x_s), s \in S, r \in R_s)$.
Φ_s	The vector $(\phi_r(x_s), s \in S, r \in R_s)$.
$J^\star$	The Jacobian of the fluid model at the equilibrium point.
$\overline{\lambda}(J^\star)$	The real part of eigenvalue of $J^\star$.
ϵ_r	The coupling window increment parameter of subflow r.
η_r	The coupling window decrement parameter of subflow r.
t	Iteration index

B. Coupled Slow-Start Algorithm

In this subsection, we introduce the Coupled SS algorithm [22] that can improve the performance of short flows' transmission using MPTCP. It consists of two parts: Coupled Ssthresh and Linked Smooth Growth. The former leverages an estimating method to reset the ssthresh value of MPTCP so that it can actively move subflows to the CA phase. The latter couples subflows and provides a slower and more friendly cwnd increment parameter, which can avoid serious packet loss caused by exponential growth. The two modules work together to adjust cwnd reasonably and achieve fairness and efficiency improvement. Moreover, Coupled SS algorithm only plays a part in the initial SS phase of MPTCP. Because when subflows of MPTCP are in different phases, it requires much extra consideration to maintain fairness with TCP, we leave it as our future work.

1) Coupled Ssthresh: Coupled Ssthresh is mainly dedicated to enabling MPTCP to quit SS timely and avoid excessive window growth and packet losses in SS. Specifically, it leverages the existing threshold estimation method (shown in detail in Section IV-B) to estimate the optimal TCP threshold. Afterward, it computes the expected throughput of the best TCP (It is the best path available to the user. In the shared bottleneck scenario, it is the path with the smallest RTT in our design.) in SS by the estimated threshold and the $\min rtt$

Algorithm 1: CSS Algorithm

1 **Initialize** $ini\_SS = true$, $\alpha = 1.5$;
2 **for** *each subflow r* **do**
3 $\quad ini\_SS_r = true$;
4 **end**
5 $con\_ssth = ssth$;
6 $exp\_th = \frac{con\_ssth}{\min rtt}$;
7 **if** *a new subflow joins* **then**
8 $\quad Totcd = \sum_{r=1} cwnd_r$;
9 $\quad NewTotcd = Totcd + IW$;
10 $\quad CT = \log_{\alpha} \frac{con\_ssth}{Totcd}$;
11 $\quad \alpha = \sqrt[CT]{\frac{con\_ssth}{NewTotcd}}$;
12 **end**
13 **if** *subflow r receives a new ACK* **then**
14 $\quad$ **if** $ini\_SS == true$ **then**
15 $\quad\quad rtt_r = g \cdot rtt_r + (1-g) \cdot rtt_r$;
16 $\quad\quad cur\_th = \sum_r \frac{cwnd_r}{rtt_r}$;
17 $\quad\quad$ **if** $cur\_th \leq exp\_th$ **then**
18 $\quad\quad\quad \alpha_r = \alpha^{\frac{rtt_r}{\min rtt}}$;
19 $\quad\quad\quad cwnd_r += \min(MSS, (\alpha_r - 1) \cdot acked\_Bytes)$;
20 $\quad\quad$ **else if** $cur\_th > exp\_th$ **then**
21 $\quad\quad\quad ini\_SS = false$
22 $\quad\quad\quad$ **for** *each subflow r* **do**
23 $\quad\quad\quad\quad ssth_r = cwnd_r$;
24 $\quad\quad\quad\quad ini\_SS_r = false$;
25 $\quad\quad\quad\quad$ `// subflows actively enter CA`
26 $\quad\quad\quad$ **end**
27 $\quad\quad$ **end**
28 $\quad$ **else**
29 $\quad\quad cwnd_r += acked\_bBytes$;
30 $\quad$ **end**
31 **end**

of subflows. To ensure fairness between TCP and MPTCP, it takes the expected throughput of the best TCP in the SS phase as the expected throughput of MPTCP. It is also a condition for MPTCP to actively move to CA. The details of the algorithm are illustrated in **Algorithm 1**.

Coupled Ssthresh algorithm firstly sets the flag of *initialSS* = *true*, and than uses the existing method to estimate the *ssth* values and *rtt* of paths available to it. Afterwards, it leverages the value of con_ssth = *ssth* and the minimum value of RTTs to calculates the expected throughput exp_th of MPTCP in SS as Line 6. Finally, the exp_{th} provides a judgment condition for MPTCP to actively quit SS. Thus, it ensures transmission efficiency of MPTCP but also guarantees fairness among MPTCP and TCP in SS phase. Specifically, CSS can compare the current throughput of MPTCP with the expected throughut, and quit SS if MPTCP has reached the expected throughput. Then, it resets the initial SS flag to false and makes each subflow enter CA without packet loss. Therefore, the algorithm ensures that MPTCP achieves the same throughput as the best single TCP at the end of SS if no packet loss occurs. When the threshold estimation is not accurate or there is packet loss in TCP SS, it still plays a role in improving fairness for reducing the gap between TCP and MPTCP, which s presented in detail in Section IV-C.

2) Linked Smooth Growth: Although the Coupled Ssthresh algorithm can guarantee fairness with TCP at the end of SS, the exponential window increment of multiple subflows may still cause a buffer overflow. To eliminate packet losses caused by the exponential growth of multiple subflows and maintain fairness with TCP during SS, CSS includes the Linked Smooth Growth algorithm. Unlike other algorithms that give fixed parameters for window increment, CSS can continuously adjust its coupling factor as a new MPTCP subflow enters. Specifically, when a new subflow is added, it calculates the sum of cwnd of all current subflows as the current total cwnd and adds IW as the new value of Total cwnd notified as *NewTotcd*. Using the given coupling factor and the calculated completion time, CSS computes the basic coupling factor α by remaining the expected completion time unchanged. In our setting, the initial value of α is 1.5 because the receiver usually uses a delayed ACK [36]. The new basic coupling factor is lower than before, which is consistent with the intuition that MPTCP should slowdown cwnd the growth of each subflow after a new subflow is added.

However, because different subflows have disparate link states, MPTCP cannot adjust its window according to the base coupling factor. CSS considers the impact of RTT to induce different window increase rates. Specifically, when the subflow receives an ACK, CSS computes the smoothed RTT value, where the parameter g is the smoothing factor whose value is 0.5. Then, it leverages the $\min rtt$ and basic parameter α obtains the coupling factor of each subflow as Line 20. Finally, according to the obtained coupling factor of each subflow, it utilizes the Byte Counting algorithm [36] to adjust the cwnd reasonably.

C. Aggressive Congestion-Control Algorithm

In this subsection, we introduce the Aggressive CA algorithm that maintains fairness between MPTCP and TCP in CA. specifically, ACA estimates the TCP throughput according to the TCP algorithm and confirms that MPTCP and TCP have the same throughput increment over the estimation period. Then it allocates total throughput increment to multiple subflows by ensuring that the traffic transfers to the subflow with better link quality. Besides, considering the throughput decrement caused by packet losses, ACA maintains the balance of increase and decrease in window adjustment to achieve a stable state. It's worth noting that ACA is also motivated by the TCP NewReno (with the Additive Increase Multiplicative Decrease scheme) as LIA and BALIA. So we only analyze and guarantee fairness with TCP NewReno connections.

Suppose there are an MPTCP and a TCP connection at a common bottleneck. Their connection information is as follows: there are *n* subflows in an MPTCP connection, the RTT value of $subflow_r$ is rtt_r, the current congestion window is $cwnd_r$, and the smallest RTT of subflows is $\min rtt$. The window of TCP connection is $cwnd^{tcp}$. If MPTCP and TCP have

the same throughput, we have:

$$\sum_{r=1}^{n} \frac{cwnd_r}{rtt_r} = \frac{cwnd^{tcp}}{\min rtt}. \tag{1}$$

For the TCP connection using Reno algorithm in the CA phase, its cwnd is increased by 1 in each RTT. As for MPTCP connections, we denote the window increment of $subflow_r$ in each rtt_r as α_r. Thus, to achieve fairness, the throughput of TCP and MPTCP satisfies

$$\sum_{r=1}^{n} \left(\frac{cwnd_r}{rtt_r} + \frac{t \cdot \alpha_r}{rtt_r} \right) = \frac{cwnd^{tcp}}{\min rtt} + \frac{t}{\min rtt} \tag{2}$$

where $cwnd^{tcp}$ is the *cwnd* of a TCP-like flow on the best available path (the one with the minimal RTT). The throughput of the best TCP can be estimated by $cwnd^{tcp}/\min rtt$. Based on SS fairness, the above formula can be simplified to

$$\sum_{r=1}^{n} \frac{\alpha_r}{rtt_r} = \frac{1}{\min rtt}. \tag{3}$$

For each period of *T*, we achieve the fairness requirement between TCP and MPTCP.

Then, we allocate the estimated throughput increase to each subflow based on its path states enabling MPTCP to remove the traffic from the congested path to the non-congested path. Firstly, the change in the throughput of a flow is proportional to the change in its congestion window size divided by its RTT

$$\triangle\ throughput \propto \frac{\triangle cwnd}{rtt}. \tag{4}$$

Thus, to share the throughput increment to all subflows reasonably, we should increase their congestion window size proportional to the RTT value observed in each route *r*. Taking two subflows $subflow_i$ and $subflow_j$ as an example, their windows are computed by

$$\frac{cwnd_i}{cwnd_j} = \frac{\alpha_i \cdot rtt_i}{\alpha_j \cdot rtt_j}. \tag{5}$$

Then, by substituting Eq. (5) into Eq. (3), ACA allocates throughput increment to MPTCP subflows. The cwnd increment factor of $subflow_r$ at each RTT is

$$\epsilon_r = \frac{cwnd_r/rtt_r}{\min rtt \cdot \sum_{k=1}^{n} \frac{cwnd_k}{rtt_k^2}}. \tag{6}$$

Next, considering throughput decrement caused by packet losses, ACA calculates the window reduction to achieve a stable state. It assumes that the packet loss rate of the subflow *r* is p_r, the increasing factor is ϵ_r, and the decreasing factor is η_r. In the process of CA, the increases and decreases of the window size must balance out, i.e., the rate of ACKs $\times$ average increase per ACK must equal the rate of drops $\times$ average decrease per drop. That is:

$$(1 - p_r) \cdot \frac{\epsilon_r}{cwnd_r} = p_r \cdot \frac{cwnd_r}{2} \cdot \eta_r. \tag{7}$$

If subflows have the same packet loss rate, making the approximation that p_r is small enough that $1 - p_r \approx 1$, it can be obtained that $cwnd^{tcp} = \sqrt{2/p_r}$. Under the fairness requirement of consistent throughput, $cwnd^{tcp}$ meets the requirements shown in Eq. (2). Substituting approximation condition and the value of ϵ_r, the value of η_r can be calculated as follow:

$$\eta_r = \frac{\min rtt \cdot \left(\sum_{k=1}^{n} cwnd_k/rtt_k\right)^2}{rtt_r \cdot cwnd_r \cdot \sum_{k=1}^{n} \left(cwnd_k/rtt_k^2\right)}. \tag{8}$$

If subflows have different packet loss rates, ACA uses the same window decrease strategy as TCP, that is, halving the window and $\eta_r = 1$; Thus, the CA algorithm can be written as

$$cwnd_r = \begin{cases} cwnd_r + \frac{\epsilon_r}{cwnd_r}, \\ cwnd_r - \frac{cwnd_r}{2} \cdot \eta_r, \end{cases} \tag{9}$$

where ϵ_r and η_r are obtained by Eq. (6) and Eq. (8), respectively. The total process is shown in **Algorithm** 2. We theoretically analyze the fairness performance of the algorithm and discuss the details of implementation in the next section.

Algorithm 2: ACA Algorithm

```
when ini_SS == false:
if subflow r receives a new ACK then
    if cwnd_r < ssth then
        cwnd_r += acked_Bytes;
    else
        cwnd_r += 1 / (rtt_r · min rtt · Σ_{k=1}^{n} cwnd_k/rtt_k^2);
    end
else if subflow r experiences packet loss then
    if subflows have the same packet loss rate then
        cwnd_r -= min rtt · (Σ_{k=1}^{n} cwnd_k/rtt_k)^2 / (2 · rtt_r · Σ_{k=1}^{n} (cwnd_k/rtt_k^2));
    else
        cwnd_r -= cwnd_r / 2;
    end
end
```

IV. Method Analysis

In this section, we analyze the fairness of TCCC and discuss the impact of ssthresh estimation and link bottleneck on congestion control algorithms. Specifically, we use the fluid model to uniformly represent the algorithms. Then, we compare the available metrics of algorithms' aggressiveness to present the fairness performance of TCCC.

A. Fluid Model

Firstly, we present the fluid model utilized to analyze the theoretical efficiency of ACA. As described in [5], the model considers a network that consists of a set $L = \{1, \ldots, |L|\}$ of links with finite capacity c_l is shared by a set of sources $S = \{1, \ldots, |S|\}$. Each source $s \in S$ is available to a fixed collection of routes (or paths) *r*, and $R := r|r \in s, s \in S$ is the collection of all routes. The routing matrix $H \in 0, 1^{|L|x|R|}$ denotes the relationship between links and routes, where $H_{lr} = 1$ if link l is in route *r*, and 0 otherwise. Each

source s maintains a congestion window $cwnd_r(t)$ and round trip time rtt_r at time t for every route $r \in s$ to calculate the sending rate $x_r(t) := cwnd_r(t)/rtt_r$. For each link l, $y_l(t) := \sum_{r\in R} H_{lr}x_r(t)$ is the aggregate traffic rate. Let $p_r(t) := \sum_{l\in L} H_{lr}p_l(t)$ represents the approximate packet loss probability. Where $p_l(t)$ denotes a congestion price at time t for link l. Thus, each route $r \in s$ associate three variables $(x_r(t), cwnd_r(t), q_r(t))$, and each source $s \in S$ can be presented by $(\boldsymbol{x}_s(t), \boldsymbol{w}(s)_t, \boldsymbol{q}_s(t))$, where $(\boldsymbol{x}_s(t) := (x_r(t), r \in s))$, $(\boldsymbol{w}_s(t) := (cwnd_r(t), r \in s))$, $(\boldsymbol{q}_s(t) := (q_r(t), r \in s))$.

Motivated by the AIMD algorithm of TCP NewReno, Peng et al. [5] model MPTCP algorithm as follows, where time t is omitted for simplicity:

$$\dot{x}_r = k_r(\boldsymbol{x}_s)(\phi_r(\boldsymbol{x}_s) - p_r)^+_{x_r} \quad r \in s \quad s \in S, \tag{10}$$

$$\dot{p}_l = \gamma_l(y_l - c_l) \quad l \in L \tag{11}$$

where $(a)^+_x = a$ for $x > 0$ and $\max\{0, a\}$ for $x \le 0$, and γ_l is a positive parameter determining the dynamic property. Suppose a source s increases its window at the return of each ACK and the increment is denoted by $I_r(\boldsymbol{w_s})$ and decreases the window on route r when a packet loss occurs and the decrement is denoted by $D_r(\boldsymbol{w_s})$ when a packet loss occurs. Then, $k_r(\boldsymbol{x_s})$ and $D_r(\boldsymbol{x_s})$ can be computed by

$$\begin{cases} k_r(\boldsymbol{x_s}) = \frac{x_r}{rtt_r} D_r(\boldsymbol{w_s}), \\ \phi_r(\boldsymbol{x_s}) = I_r(\boldsymbol{w_s})/D_r(\boldsymbol{w_s}). \end{cases} \tag{12}$$

The MPTCP algorithm installed at source s can be specified by $(\boldsymbol{K_s}, \boldsymbol{\Phi_s})$, where $\boldsymbol{K_s}(\boldsymbol{x_s}) := (k_r(\boldsymbol{x_s}), r \in s)$ is a vector of positive gains that determines the dynamic properties of the algorithms, and $\boldsymbol{\Phi_s}(\boldsymbol{x_s}) := (\phi_r(\boldsymbol{x_s}), r \in s)$ determines the equilibrium properties of the algorithms. Beside, the shape of vectors $(\boldsymbol{K_s}, \boldsymbol{\Phi_s})$ also rank the different variants with respect to their friendliness and responsiveness (For the full details, see [5]). More specifically:

Theorem 1 (Friendliness): Under certain assumptions (which are intuitive and usually (but not always) satisfied), an MPTCP algorithm $(\tilde{\boldsymbol{K}}, \tilde{\boldsymbol{\Phi}})$ is friendlier than another algorithm $(\hat{\boldsymbol{K}}, \hat{\boldsymbol{\Phi}})$ if $\tilde{\boldsymbol{\Phi}}_{\boldsymbol{s}}(\boldsymbol{x_s}) \le \hat{\boldsymbol{\Phi}}s(\boldsymbol{x_s})$.

Theorem 2 (Responsiveness): For two MPTCP algorithms, if one has a smaller eigenvalue real part of Jacobian $J^\star$

$$\bar{\lambda}(J^\star) = \max\left\{ \frac{\boldsymbol{x}^H \left[\frac{\partial \boldsymbol{\Phi_s}}{\partial \boldsymbol{x_s}}\right]^+ \boldsymbol{x}}{\boldsymbol{x}^H \Lambda_k^{-1} \boldsymbol{x} + \boldsymbol{p}^H \Lambda_\gamma^{-1} \boldsymbol{p}} \right\}, \tag{13}$$

it is more responsive, where $\Lambda_k = diag\{k_r(x_s), r \in R\}$, $\Lambda_\gamma = diag\{\gamma_l . r \in L\}$, and $\frac{\partial \boldsymbol{\Phi_s}}{\partial \boldsymbol{x_s}}$ are evaluated at the equilibrium point. The theorem can be simplified as: An MPTCP algorithm with a larger $\boldsymbol{K_s}(\boldsymbol{x_s})$ and more negative definite $\frac{\partial \boldsymbol{\Phi_s}}{\partial \boldsymbol{x_s}}$is more responsive.

B. Fairness Analysis

In this part, we provide fairness analysis of our scheme that includes ACA and CSS, respectively.

TABLE II
MODEL PARAMETERS OF SEVERAL REPRESENTATIVE ALGORITHMS

Algorithm	$k_r(x_s)$	$\phi_r(x_s)$
Reno	$\frac{1}{2}x_r^2$	$\frac{2}{rtt_r^2 x^2}$
EWTCP	$\frac{1}{2}x_r^2$	$\frac{2a}{rtt_r^2 x^2}$
Coupled	$\frac{1}{2}x_r^2$	$\frac{2}{rtt_r^2(\sum_{k\in s} x_k)^2}$
BALIA	$\frac{x_r \min x_r}{2}$	$\frac{2}{rtt_r^2(\sum_{k\in s} x_k)^2} \cdot \frac{4+5\alpha_r+\alpha_r^2}{10\alpha_r}$
e-NMCC	$\frac{x_r^2(\sum_{k\in s} x_k)^2}{2\sum_{k\in s} x_k^2}$	$\frac{2\max_{k\in s}(1/rtt_k^2)\sum_{k\in s} x_k^2}{x_r^2 rtt_r^2 \sum_{k\in s} 1/rtt_k^2 (\sum_{k\in s} x_k^2)^2}$
TCCC	$\frac{x_r \cdot \min rtt \cdot (\sum_{k\in s} x_k)^2}{2 \cdot rtt_r^2 \cdot \sum_{k\in s} cwnd_k/rtt_k^2}$	$\frac{2}{\min rtt^2 \cdot (\sum_{k\in s} x_k)^2}$

1) ACA Analysis: We use the fluid model proposed by Peng [5] to analyze the theoretical efficiency of ACA. We take TCP-NewReno as an example to show the function of this model. As TCP-NewReno adjusts the window as

- For each ACK on route r, $cwnd_r \leftarrow cwnd_r + \frac{1}{cwnd_r}$,
- For each loss on route r, $cwnd_r \leftarrow \frac{cwnd_r}{2}$,

its model coefficients can be expressed as

$$k_r(x_s) = \frac{1}{2}x_r^2, \quad \phi_r(x_s) = \frac{2}{rtt_r^2 x^2}. \tag{14}$$

When subflows have the same packet loss rate, according to the definition of the model, the window increment and decrement rates of TCCC can be expressed as

$$I_r(w_s) = \frac{1/(rtt_r \cdot \min rtt)}{\sum_{k\in s} x_k/rtt_k},$$

$$D_r(w_s) = \frac{\min rtt \cdot \left(\sum_{k\in s} cwnd_k/rtt_k\right)^2}{2rtt_r \cdot \sum_{k\in s} cwnd_k/rtt_k^2}. \tag{15}$$

Thus, the corresponding performance parameters are

$$k_r(x_s) = \frac{x_r \cdot \min rtt \cdot \left(\sum_{k\in s} x_k\right)^2}{2 \cdot rtt_r^2 \cdot \sum_{k\in s} cwnd_k/rtt_k^2},$$

$$\phi_r(x_s) = \frac{2}{\min rtt^2 \cdot \left(\sum_{k\in s} x_k\right)^2}. \tag{16}$$

For readers to clearly follow the fairness analysis of the ACA algorithm, we also give the model parameters of several representative algorithms in **Table II**.

To take advantage of known theories, we prove that TCCC satisfies the required constraints and verified that TCCC satisfies the required conditions (The specific details of conditions from C0 to C5 are shown in [5].) as Coupled. Then, we prove the friendliness and responsiveness performance of the scheme according to the given theorems.

Firstly, we present "friendliness" performance of algorithms by comparing values of Φ_r:

$$\phi_r^{Coupled} \le \phi_r^{BALIA} < \phi_r^{EWTCP} < \phi_r^{TCP},$$
$$\phi_r^{Coupled} \le \phi_r^{TCCC} < \phi_r^{EWTCP} < \phi_r^{TCP}. \tag{17}$$

It can be easily deduced that when there are n subflows in MPTCP, the order of the Φ value of algorithms is consistent with the above. From the Theorem 1, we can prove that TCCC and BALIA all behaves no better friendly than Coupled.

Next, we compare the "responsiveness" performance of different algorithms. We suppose that all subflows of MPTCP have the same RTT, then we obtain that

$$k_r(\boldsymbol{x}_s)^{TCCC} = \frac{x_r\left(\sum_{k\in s} x_k\right)^2}{2\left(\sum_{k\in s} x_k\right)} \geq \frac{1}{2}x_r^2. \tag{18}$$

Thus, for each s, we deserve:

$$\boldsymbol{K_s}^{TCCC} \geq \boldsymbol{K_s}^{BALIA} \geq \boldsymbol{K_s}^{Coupled}. \tag{19}$$

By comparing the $\boldsymbol{K_s}$ value and $\tilde{\boldsymbol{\Phi}}_{\boldsymbol{s}}$ of different algorithms, we prove that TCCC is more responsive than Coupled based on the Theorem 2. However, when compared with BALIA and EWTCP, we cannot directly achieve responsiveness performance according to the simplified theorem because their relation of $\boldsymbol{K_s}$ and $\boldsymbol{\Phi_s}$ are staggered. Thus, we utilize the basic "Responsiveness" theorem to compare their $\lambda(\overline{J}^{\star})$. In the special cases where all states of MPTCP subflows are equal, we have

$$\lambda\left(\overline{J}^{\star}\right)^{BALIA} \leq \lambda\left(\overline{J}^{\star}\right)^{TCCC} \leq \lambda\left(\overline{J}^{\star}\right)^{TCP}. \tag{20}$$

Therefore, we can be obtained that TCCC is more aggressive than BALIA, but it still does not exceed TCP. The same conclusion can also be proven when subflows have different packet loss rates. Thus, we can conclude that TCCC maintains fairness with TCP.

We must note that, the response to packet loss of the e-NMCC algorithm is more involved than the model prescribes and our analysis is carried out under special conditions. In the next section, we present the performance through detailed simulation testing and actual tests in our testbed for more specific measurements of the algorithm.

2) CSS Analysis: As the SS window adjustment follows a simple formula, we prove its fairness performance by mathematical deduction. We assume that there are a TCP connection and an MPTCP connection with n subflows competing bottleneck bandwidth. The ssthresh value of TCP and MPTCP's subflow_r obtained by the estimation algorithm is *ssth* and $ssth_r$, respectively. The corresponding RTT is *rtt*, rtt_r, and $\min rtt$ denotes the minimum *rtt* of all paths.

As for the uncoupled SS algorithms (e.g., LIA and BALIA), their maximal throughput in SS is $\sum_{r=1}^{r=n} \frac{ssth_r}{rtt_r}$ at the end of SS, while the throughput of TCP is $\frac{ssth}{rtt}$. If rtt_r is the same as *rtt*, it is reasonable to deduce that $ssth_r$ equals *ssth*. Thus, MPTCP algorithms with uncoupled SS have n times of throughput at the end of SS if the estimation of ssthresh is accurate. If these algorithms don't use the threshold estimation method and set the ssthresh to its default value. Algorithms usually suffer packet loss before reaching the value. MPTCP algorithms with uncoupled SS still have a higher window increase rate compared with TCP. While in CSS, it utilizes $con\_ssth = \max\{ssth_r\}$ to actively enter the CA phase when it achieves the expected throughput. Thus, the throughput of CSS is $\frac{con\_ssth}{\min rtt}$, which equals the estimated throughput of the best TCP connection. Besides, CSS continuously adjusts the coupling factor of each subflow as a new MPTCP subflow enters maintaining fairness during SS.

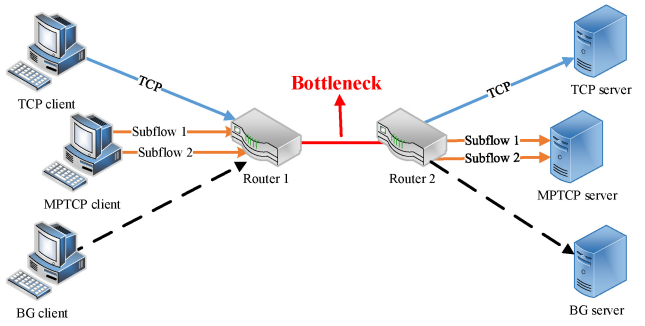
C. Ssthresh Estimation

In TCP, the SS phase aims to discover the available network bandwidth effectively, and the ssthresh is the upper bound of cwnd to prevent serious packet loss caused by burst data. However, the SS ssthresh is still a fixed value in the existing network environment, which is highly likely to cause continuous packet loss and performance degradation for TCP transmission. There is some work devoted to finding a proper ssthresh to solve the problem of continuous packet loss caused by fixed values in TCP connection. In [37], Hoe et al. revealed that data packets sent closely spaced may arrive at the receiver at the rate of the bottleneck link bandwidth. Thus, they proposed to calculate an approximation of the ssthresh by using the least-squares estimation on three closely-spaced ACKs received at the sender and their respective time of receipt. Zhang et al. [32] proposed to adjust the value of parameter ssthresh dynamically according to the bandwidth, packet loss, and current RTT. With a reasonable approximation, our scheme can prevent the excessive window increase that leads to multiple packet losses. Thus, it may improve transmission reliability, reduce transmission delay, and ensure friendliness among TCP and MPTCP.

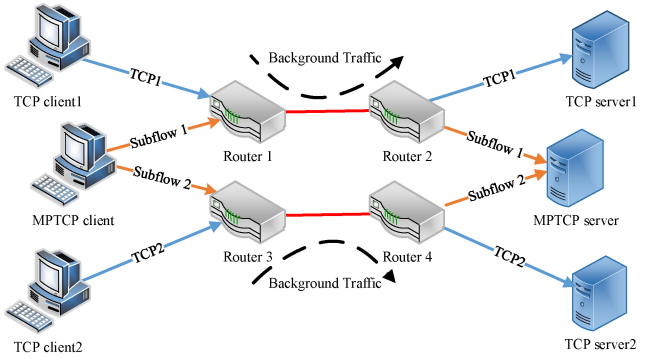
Next, we discuss the impact of estimation accuracy, which may decrease with the dynamic changes of networks, on CSS algorithms. If the maximum throughput limitation is too low, the sender would prematurely switch to the additive increase mode, and the transfer time would be longer. If the approximation is too high, MPTCP and TCP both suffer packet loss before reaching the ssthresh value. In this case, compared with those decoupled schemes, our algorithm can still play a role in reducing burst data for slowing down the window increase rate. Therefore, we use the average estimated value when the network is stable and choose a larger estimation value to ensure transmission efficiency in dynamic networks.

D. Bottleneck Detection Problem

When the subflows of MPTCP do not share a bottleneck, using the principle of network fairness to couple the growth of congestion window will significantly reduce the transmission efficiency of MPTCP. To achieve higher algorithm efficiency, we utilize the existing bottleneck detection schemes to detect bottlenecks, and then decouple the growth rate of subflows when they do not share bottlenecks. We investigated the existing related works in bottleneck detection in detail and find three reliable solutions: DWC [38], MPTCP-SBD [39], and SB-CC [35]. They detect bottlenecks by observing delay increase and packet loss during the same period, specifically, comparing three key statistics of one-way delay, leveraging packet loss signals during the observation window, and using ECN signals, respectively. Among them, the SB-CC strategy is more reliable and more adaptable to dynamic networks than others. Besides, SB-CC provides a secondary judgment process to ensure detection accuracy. Thus, in scenarios that support ECN, we integrate our scheme with SB-CC to improve the efficiency of MPTCP under shared bottleneck and non-shared bottleneck scenarios. However, how determining whether to use the coupling algorithm in the SS



(a) shared Bottleneck Scenario



(b) non-shared Bottleneck Scenario

Fig. 1. Topologies used for TCCC evaluation.

is still an unsolved problem. MPTCP is prone to packet losses due to exponential growth of cwnd in SS when the buffer size is small. Thus, we use a coupled algorithm in SS at the beginning of the connection and change to an uncoupled algorithm when detected that subflows do not share a bottleneck.

V. Performance Evaluation

In this section, we evaluate the effectiveness of TCCC on short flows and long flows through the NS3 simulator and a real testbed implemented with the Linux kernel of MPTCP. In addition, the algorithms we used for performance comparison are mainly LIA, LISA, BALIA, and e-NMCC. As in [3] and [5], all TCP connections use the TCP NewReno algorithm. In order to ensure the accuracy of data, we run each experiment 50 times and present the average result and its standard deviation. Next, we show the performance of TCCC in simulation and the real platform, respectively.

A. Setup

First, we introduce the experimental environment topology and related configurations in the two scenarios.

1) Simulation Setup: We evaluate our mechanism proposed on NS3 simulator [40] with the MPTCP NS3 code provided by google MPTCP group [41]. We consider two experimental scenarios: the shared bottleneck scenario as shown in Fig. 1(a) and the non-shared bottleneck scenario as shown in Fig. 1(b). In the shared bottleneck scenario, there is an MPTCP connection with two subflows that shares a common bottleneck with a TCP connection. Specifically, MPTCP client and MPTCP server are both multi-homed, and they establish two subflows between them. The TCP client and TCP server are single-homed and they share a bottleneck from Route1 to Route2 with MPTCP connection. BG client and BG server are to simulate background traffic. Its bandwidth occupation fluctuation from 1% to 5% of the bottleneck bandwidth to simulate dynamic networks.

In the non-shared bottleneck scenario, two subflows of MPTCP travel through different bottlenecks independently, and each subflow competes for bandwidth with one TCP connection. The background traffic is generated the same as the shared bottleneck scenario. Moreover, the bandwidth at the bottleneck is set to 5Mbps and the delay is 30ms. The bandwidth of other paths is 100Mbps and the delay is 10ms. The buffer size at the bottleneck is set to 47 packets, which equals the Bandwidth-Delay Product of the path. Besides, the packet size is set to 1400 bytes in our simulations.

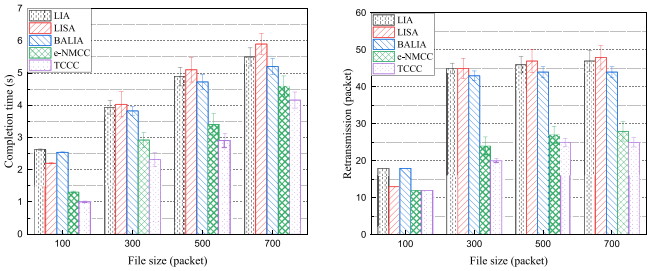
2) Testbed Setup: The real testbed is used to present the performance of TCCC in real networks. We focus on the shared bottleneck scenario and utilize the same topology as the simulation shown in Fig. 1(a). The testbed includes eight PCs running on 64 bits Ubuntu 16.04, and their roles are as follows: one MPTCP client, one TCP client, two routers, two corresponding servers using Apache2 [42], and two clients generate background traffics. Both MPTCP client and MPTCP server are implemented with the MPTCP 0.94 [43]. Background traffic is a TCP connection made by iperf [44]. We use wget to download files with a specific size and compare the average download time and bandwidth share of MPTCP in the bottleneck. Besides, The link information is the same as that in the software simulation.

B. NS3 Simulation

In this subsection, we show how TCCC performs in the ns3 simulation environment. In this scenario, we simulate short flows and long flows by file transfer. Specifically, we transfer files with a number of packages less than 700 to simulate short flows and transfer files that are larger than 10000 packets to simulate long flows. We consider both shared bottleneck and non-shared bottleneck situations and present the efficiency of algorithms with varying buffer sizes and file sizes.

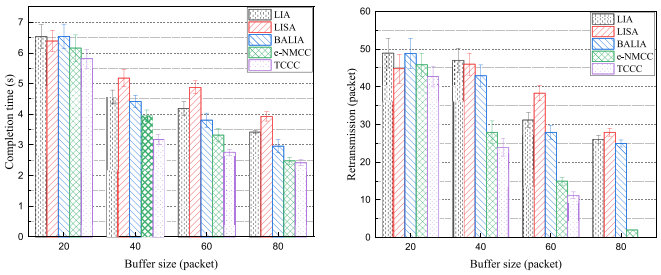
1) Short Flows: We first show the performance of TCCC for short flows by downloading small files (less than 1MB). We consider both the impact of file size and buffer size of the bottleneck on transmission efficiency. Next, we present the performance under shared bottleneck and non-shared bottleneck scenarios, respectively.

Shared Bottleneck Scenario: To show the performance of our algorithm on short flows in the shared bottleneck scenario, we set the bottleneck buffer size to 47 packets and download files of different sizes from 100 packets (140KB) to 700 packets (980KB) with an interval of 200 packets between each one. Fig. 2(a) shows the average file completion time and Fig. 2(b) shows the corresponding retransmissions. We can observe that by coupling SS and slowing down the growth rate of the congestion window, TCCC suffers less packet loss and completes the file transfer within a shorter time. Specifically, compared with LIA, TCCC reduces the download time by

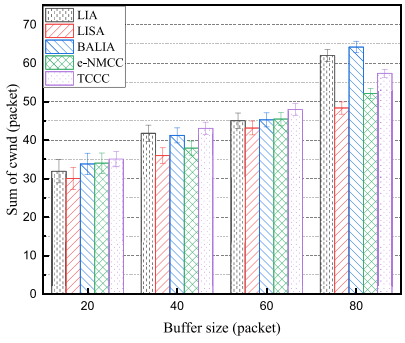


(a) Influence of file size on completion time. (b) Retransmissions of fixed buffer size.

Fig. 2. File transfer with fixed buffer size (47 packets) in the shared bottleneck scenario.



(a) Influence of buffer size on completion time. (b) Retransmissions of changing buffer size.



(c) *Sum of cwnd* when completing transfer.

Fig. 3. File transfer with fixed file size (400 packets) in the shared bottleneck scenario.

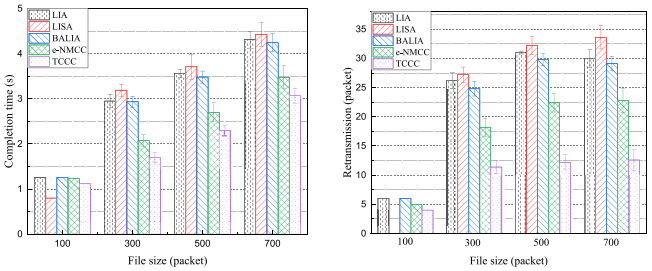
0.5s on average and 2.2s in the best case. It also reduces the download time by 0.4s compared with e-NMCC. When the number of packets is smaller than 100, the transmission completion time of LISA is less than that of LIA and BALIA, since LISA removes the excessive increase of new-added subflow. As the file size increases (larger than 100 packets), LISA performs badly because it is unable to solve packet losses caused by exponential window increments.

Since the buffer size of the bottleneck can significantly affect the performance of short flows, we change the buffer size at the bottleneck to evaluate the adaptability of TCCC in various network environments. Specifically, we vary the buffer size from 20 to 80 packets (200% of the path's BDP) and download the file size of 400 packets (560KB). The average file completion time and the corresponding retransmissions are shown in Fig. 3(a) and Fig. 3(b), respectively. It can be observed that with the increase in buffer size, the completion time of all algorithms decreases. This is because a larger buffer can accommodate more burst data and reduce the possibility of packet loss. Thus, with a larger buffer size, these algorithms can reduce the number of retransmitted packets and complete the transmission earlier. When the buffer size is 20 packets, the completion time of all algorithms is larger, since all the algorithms suffer more packet loss than in the case where the buffer is small. In this case, TCCC still performs best and it reduces the completion time by about 1s compared with the worst one. When the buffer size exceeds 40 packets, TCCC can drastically reduce packet loss, and continuously bring a goodput gain over LIA and BALIA, e.g., up to 40% at the buffer size of 60 packets. From Fig. 3(b), we can observe that the number of retransmissions of TCCC decreases significantly with the increase in buffer size. When the buffer varies from 40 packets to 60 packets, the number of retransmitted packets of TCCC is reduced by 57.45%. When the buffer size is 80 packets, TCCC experiences no packet loss. Therefore, its completion time reaches the minimum. These minimums are caused by their upper bounds on total cwnd, i.e., the ssthresh and con_ssth.

In order to avoid buffer overflow at the bottleneck, TCCC sets con_ssth to exit SS in advance and enter the CA phase smoothly. However, this may lead to throughput reduction of MPTCP when it exits SS with a too-small cwnd. To evaluate the impact of this tradeoff, we test how the Sum of cwnd of MPTCP algorithms varies with the buffer size at the bottleneck after 400 packets (560kb) are transmitted. As shown in Fig. 3(c), with the increase of buffer size, the Sum of cwnd of all algorithms increases. Specifically, when the buffer size varies from 20 packets to 80 packets, the Sum of cwnd of TCCC, e-NMCC, LIA, and TCP increases by 30.77%, 24.14%, 36.73%, and 45.45%, respectively. When the buffer size increases from 20 packets to 60 packets, the Sum of cwnd of TCCC is still the largest of all algorithms. This is because it couples the increment of cwnd and reduces the packet loss in transmission. Therefore, the window drops down caused by packet loss is less than that of other algorithms, which results in a larger final Sum of cwnd. However, when the window size is 80 packets, TCCC is inferior to LIA and BALIA because the larger buffer can also reduce packet loss. Although TCCC does not perform well when the buffer size is too large, it is still better than e-NMCC, and it is best when the buffer is small.

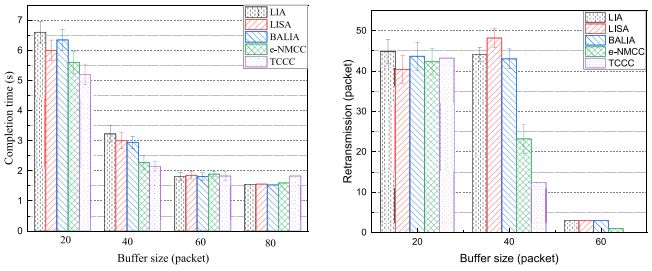
Non-shared Bottleneck Scenario: In this part, we present the performance of different algorithms in a non-shared bottleneck scenario. At first, we fix the buffer size from Route1 to Route2 to 47 packets(one BDP) and vary the file size from 100 packets (140KB) to 700 packets (980KB). Fig. 4(a) and Fig. 4(b) show the average completion time and the corresponding retransmissions, respectively. We observe that TCCC performs better and reduces the completion time by about 1.2s on average as compared with other algorithms, which is mainly due to the decrease in retransmissions. Specifically, TCCC reduces the transmission time of the file with 300 packets by 42.37%, 42.35%, 46.71%, and 17.87% as compared with LIA, BALIA, LISA, and e-NMCC, respectively.

To evaluate the adaptability of TCCC in a non-shared bottleneck scenario, we vary the bottleneck buffer size from

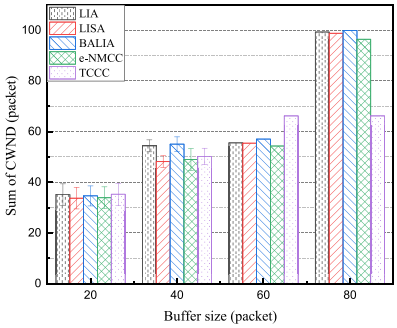


(a) Influence of file size on completion time. (b) Retransmissions of fixed buffer size.

Fig. 4. File transfer with fixed buffer size (47 packets) in the non-shared bottleneck scenario.



(a) Influence of buffer size on completion time. (b) Retransmissions of changing buffer size.



(c) *Sum of cwnd* when completing transfer.

Fig. 5. File transfer with fixed file size (400 packets) in the non-shared bottleneck scenario.

20 packets to 80 packets to download the file with the size of 400 packets (560KB). Fig. 5(a) shows the average completion time and Fig. 5(b) shows the corresponding retransmissions. With the increase in buffer size, the number of retransmitted packets of TCCC decreases significantly. When the buffer size is smaller than 60 packets, TCCC reduces packet loss and brings a goodput gain of up to 45.35% as compared to LIA. When the buffer size exceeds 60 packets, TCCC is inferior to other algorithms. The reason is the same as in a shared bottleneck scenario with a large buffer size. We can also conclude that the buffer size of the bottleneck also has a significant effect on the transmission of short flows in a non-shared bottleneck scenario.

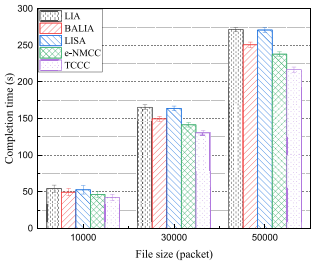
Though the con_ssth is good for short flows, it may hurt the performance of MPTCP for long flows when its subflows do not share a bottleneck. We present how the Sum of cwnd varies with the buffer size at the bottleneck after completing the transfer of 400 packets (560KB). As shown in Fig. 5(c), when the buffer size is smaller than 60 packets, TCCC has a larger Sum of cwnd than other algorithms. That means, TCCC can still improve the transmission efficiency of long streams when the buffer size is less than 60 packets. However, when the buffer size exceeds 60 packets, the buffer is large enough to prevent LIA, BALIA, and LISA from packet loss during SS. As a result, these algorithms perform better than TCCC for a higher cwnd. Although the performance of TCCC is not ideal when the buffer size is large, it still performs best when the buffer size is very small and this problem can be solved by accurate threshold detection.

We can conclude that using coupled SS can effectively reduce packet loss caused by burst data and improve the transmission efficiency of short flows when the buffer is small. We also demonstrate that the buffer size has a significant impact on the transmission of short streams. TCCC not only performs well in shared bottleneck scenarios but also works in non-shared bottleneck scenarios when the buffer size is smaller than BDP. Specifically, TCCC can reduce the transmission time by about 1.5s compared with LIA and BALIA, and its completion time is 17.87% lower than e-NMCC at the best time.

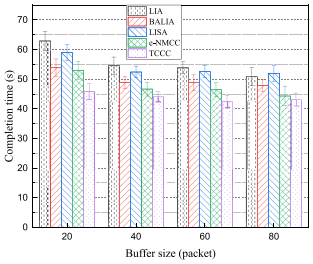
2) Long Flows: Then, we show the effect of our algorithm on long flows by downloading larger files (larger than 10MB). We also consider the impact of different file sizes and buffer sizes on transmission efficiency. As there are only two flows (MPTCP flow and TCP flow) competing bottleneck bandwidth in our testing, we show the fairness performance of algorithms by comparing the bandwidth occupation ratio of each algorithm to TCP. Although the presentation pattern is different, the result is consistent with that of using Jain's fairness index [45]. Moreover, we evaluate the algorithm performance in both the shared Bottleneck and non-shared Bottleneck scenarios.

Shared Bottleneck Scenario: Similar to tests on short flows, we first present the performance of MPTCP algorithms in the shared bottleneck scenario. We fix the size of the buffer to 47 packets and download the data of different sizes from 10000 packets (14MB) to 50000 packets (70MB), and the result is shown in Fig. 6(a). By comparing the average file completion time, We can deserve that TCCC with a more aggregative increasing factor is superior to other algorithms and completes the transmission earlier. Specifically, TCCC reduces the transmission time by 28.83% as compared with LIA, and by 9.77% with e-NMCC in downloading 10000 packets situation.

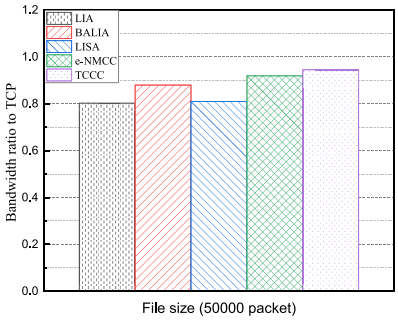
Then, to show the impact of buffer size on the long flows, we adjust the buffer size from 20 to 80 packets to download the same data with 10000 packets. As can be seen from Fig. 6(b), the effect of buffer size on the completion time of long flows is less significant than that of short flows. When the buffer is larger than 40 packets, the completion time does not decrease with the increase in the buffer size, because the window adjustment has reached a relatively stable state in long flows. When the buffer is 20 packets, the completion time of each algorithm is larger than that of other buffer sizes because of burst data. Thus, when the buffer is limited, long flows will also suffer packet loss and retransmissions. Anyway, TCCC still has



(a) Influence of file size on completion time.

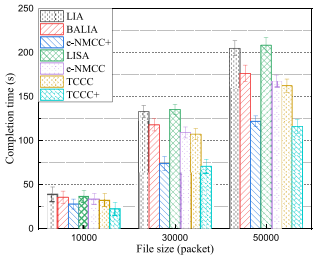


(b) Influence of buffer size on completion time.

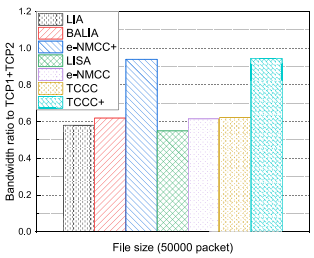


(c) The bandwidth ratio to TCP during file transfer.

Fig. 6. The performance of long flows in the shared bottleneck scenarios.



(a) Influence of file size on completion time.



(b) Bandwidth ratio to TCPs during file transfer.

Fig. 7. The performance of long flows in the non-shared bottleneck scenario.

less completion time than LIA, BALIA, and e-NMCC under buffer changing scenarios. We take a buffer size of 40 packets as an example: TCCC reduces the transmission time by 28.57% compared with LIA, by 25.7% compared with LISA, and by 14.67% compared with e-NMCC.

To verify the fairness of MPTCP algorithms, we compare the bandwidth ratio of a single TCP and MPTCP. In this situation, TCP starts to transmit at the beginning of the simulation like MPTCP and ends at the same time when MPTCP completes the transmission to fairly compare the performance of TCP and MPTCP algorithms. Besides, we compare the bandwidth ratio of each MPTCP algorithm compared with TCP, where we take the average throughput of all algorithms under test as their bandwidth and then calculate the bandwidth proportion. From Fig. 6(c) we can see that TCCC's bandwidth share is not more than TCP, but the bandwidth share of TCCC is improved compared with other algorithms. It is because other algorithms restrict the increment of cwnd in CA to satisfy the second design goal of MPTCP congestion control. Therefore, TCCC can improve the competitiveness of MPTCP and make full use of bandwidth resources without affecting the efficiency of TCP.

In general, for long flow transmission, TCCC performs well at the shared bottleneck in the simulation environment. Through the relevant test, we can conclude that the more aggressive window adjustment factors of TCCC in the CA phase can improve its competitiveness at the bottleneck. Compared with LIA and BALIA, it can improve the transmission efficiency of MPTCP at the bottleneck based on ensuring fairness with TCP. Moreover, different from the transmission of short flows, buffer size has little effect on long flow transmission.

Non-shared Bottleneck Scenario: In this part, we present the efficiency of our algorithm in the non-shared bottleneck scenario. We still consider the impact of different file sizes and buffer sizes and present the fairness of our algorithm by comparing the bandwidth ratio of a single TCP. Besides, TCCC+ and e-NMCC+ represent the algorithm that uses coupled algorithms in SS and decouples in CA, which aims to present the effect of algorithms with bottleneck detection mechanisms.

First of all, we fix the size of the buffer to 100 packets (one BDP) and study the efficiency of algorithms by downloading different sizes of data. We download files with the size of 10000 packets to 50000 packets and repeat the experiment fifty times to obtain the average completion time as the final download time. As shown in Fig. 7(a), TCCC can still effectively reduce the file download time in the non-shared bottleneck scenario. Specifically, TCCC can reduce file download time by 13.13% compared with LIA when the file size is 30000 packets. Moreover, in the sharing bottleneck scenario, the gap between TCCC, and LIA is smaller and LIA performs better than LISA. For example, when the file size is 30000 packets, the gap between LIA and TCCC is reduced from 31.55% to 13.13% compared with the sharing bottleneck. This is because the coupled SS makes the window growth slower than the uncoupled SS algorithms, while the packet loss caused by burst data in the non-shared bottleneck scenario is less than that in the shared scenario.

To show the results after using bottleneck detection algorithms, we still use the coupled SS for TCCC and e-NMCC, but we assume that the bottleneck detection is completed after the subflow enters the CA phase, and then uses the decoupled window adjustment method. The TCCC and e-NMCC using the decoupled algorithm if subflows do not share a bottleneck after bottleneck detection is called TCCC+ and e-NMCC+. We can see that TCCC+ and e-NMCC+ can greatly reduce the completion time of files compared with LIA and BALIA, and TCCC+ performs better than e-NMCC+. Besides, when the file size is fixed and the buffer size is gradually increased, we observe that similar to the shared bottleneck scenario, the impact of buffer size on the average completion time of a file is relatively small, so we do not present the results anymore.

We also study the fairness performance of MPTCP algorithms in non-shared bottleneck scenarios. By comparing the bandwidth value of MPTCP with that of TCP flows, we calculate the bandwidth ratio of MPTCP algorithms relative to the sum of TCP1 and TCP2. As shown in Fig. 7(b), the coupled

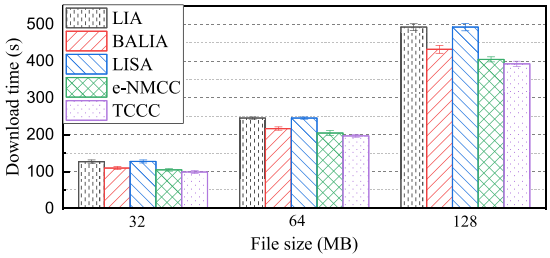


Fig. 8. Long flows Performance in shared bottleneck under our testbed.

TABLE III
BANDWIDTH RATIO OF MPTCP TO TCP USING WGET TESTING IN SHARED BOTTLENECK SCENARIO

Algorithm	LIA	BALIA	LISA	e-NMCC	TCCC
Average Ratio	0.804	0.937	0.802	0.962	0.979

congestion algorithms will seriously damage the efficiency of MPTCP when the subflows are not sharing bottlenecks. Coupled SS with decoupled CA can still effectively improve the efficiency of MPTCP in non-shared bottleneck scenarios.

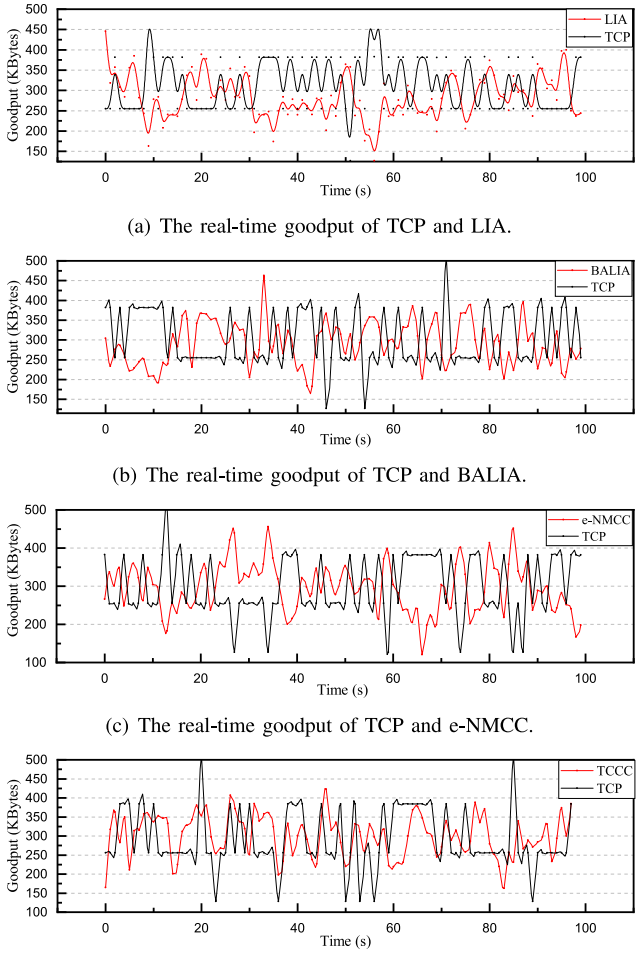
C. Real Testbed

In this part, we present the performance of TCCC in our testbed, where we consider two aspects including file completion time and real-time transfer rate. First, we show the transmission efficiency of TCCC under the shared bottleneck scenario, and the topology is shown in Fig. 1(a). In the non-shared bottleneck scenario, we consider an asymmetric environment shown in Fig. 10, where two subflows of MPTCP have a large difference in transmission delay.

Firstly, We use wget to download files of different sizes from 32MB to 128MB and present the mean download time and corresponding bandwidth ratio during transmission, respectively. From Fig. 8, we can see that TCCC and e-NMCC can effectively reduce the download time as compared with LIA and BALIA. Specifically, when downloading 128MB, TCCC can reduce the time by 20.16% and 9.05% as compared with LIA and BALIA, and it can also bring a better goodput by 2.97% as compared with e-NMCC.

To analyze the fairness of algorithms in the transmission process, we record the bandwidth during the transmission of 128MB and show the bandwidth ratio to TCP of each algorithm in Table III. We can see that, in the whole transmission process, LIA can only get 80.4% bandwidth compared with TCP. And its weak competitiveness at the bottleneck is the main reason for the longer transfer time. TCCC, whose bandwidth ratio value is 0.9799, ensures fairness with TCP sharing the same bottleneck while reducing the transmission time compared with other algorithms. Thus, we can conclude that TCCC can significantly improve the transmission efficiency compared with LIA and BALIA for the transmission of long flows.

In addition to comparing the achieved bandwidth at the end of file transmission, we also use iperf to initiate both MPTCP and TCP connections at the same time and record the average bandwidth per second in a duration of 100s to present



(a) The real-time goodput of TCP and LIA.

(b) The real-time goodput of TCP and BALIA.

(c) The real-time goodput of TCP and e-NMCC.

(d) The real-time goodput of TCP and TCCC.

Fig. 9. Fairness performance of different MPTCP congestion algorithms at the bottleneck within 100s.

the bandwidth changes of each algorithm. From Fig. 9 we can see that the bandwidth of LIA is lower than that of TCP until 75s, and BALIA can reach almost the same bandwidth as TCP in 12s. As for e-NMCC and TCCC, TCP connection has no obvious advantage at the beginning of transmission. By comparing the curves of TCCC and e-NMCC, we deserve that TCCC is smoother than e-NMCC. Thus, we can conclude that in the real testbed, TCCC can not only ensure fairness with TCP at the bottleneck but also has stronger competitiveness and effectively reduces the file transfer time. It also achieves fairness faster than LIA and BALIA and has a smoother window adjustment compared with e-NMCC. We repeat the tests twenty times to present the fairness performance of each algorithm. Specifically, we calculate the average value and fluctuations of the throughput ratio obtained by MPTCP relative to that obtained by TCP, which is shown in Table IV. We can obtain that TCCC performs best in fairness.

Finally, to further demonstrate the usability of TCCC in a practical environment, we also consider the performance of MPTCP when two subflows are heterogeneous. We study the performance of algorithms in the asymmetric scenario, where we utilize the topology as shown in Fig. 10. The link conditions of the two subflows are as follows: the bandwidth of

TABLE IV
BANDWIDTH RATIO OF MPTCP TO TCP USING IPERF TESTING IN SHARED BOTTLENECK

Algorithm	LIA	BALIA	e-NMCC	TCCC
Average Ratio	0.883	0.945	0.983	1.017
Ratio Volatility	±0.026	±0.033	±0.041	± 0.013

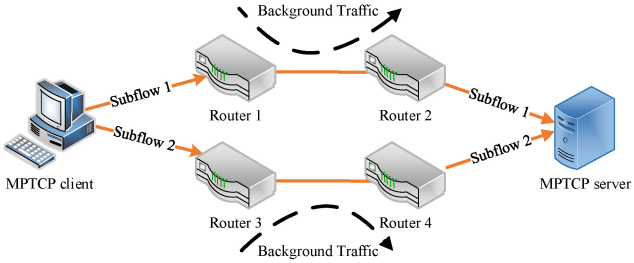


Fig. 10. The asymmetric testing topology.

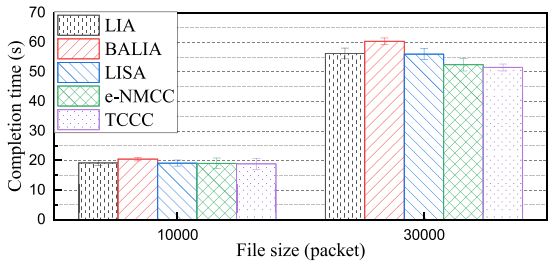


Fig. 11. The performance of algorithms in the asymmetric scene.

subflow1 is 5Mb and the latency is 10ms, while those of subflow2 are 10Mb and 50ms. We download the files of sizes from 10000 packets to 30000 packets and record the average completion time. From Fig. 11, we can perceive that BALIA, which performs better than LIA on fairness performance with TCP, performs poorly in the asymmetric scenario. For the transmission of 30000 packets, it takes about 10s more than TCCC to complete the transmission. Compared with LIA, BALIA, and e-NMCC, TCCC still improves the algorithm efficiency in the non-shared bottleneck scenario, and it still performs well in asymmetric paths.

VI. CONCLUSION

In this paper, we considered the fairness and responsibility of MPTCP for both short and long flows. We proposed a throughput consistency congestion control method named TCCC, which consists of CSS and ACA algorithms. CSS couples the window adjustment of subflows to reduce the burst data caused by the exceptional increment of cwnd in SS and resets the Ssthresh to safely enter the CA phase. ACA computes a more aggressive window adjustment factor in the CA phase to compete for more bandwidth on the condition of ensuring fairness. By testing the performance of short flows and long flows, we show the performance of TCCC in both shared bottleneck and non-shared bottleneck scenarios in software simulation and our testbed. Our results verify that for both long flows and short flows, wherever they are in shared bottleneck or non-shared bottleneck scenarios, TCCC ensures fairness and high throughput for MPTCP.

REFERENCES

[1] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 266–277, 2011.

[2] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. USENIX Symp. Netw. Syst. Des. Implement. (NSDI)*, vol. 11, 2011, pp. 99–112.

[3] C. Raiciu, M. J. Handley, and D. Wischik, "Coupled congestion control for multipath transport protocols," IETF, RFC 6356, 2011. Accessed: Jan. 2023. [Online]. Available: https://www.ietf.org/rfc/rfc6356.txt

[4] 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.

[5] 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.

[6] A. M. Abdelmoniem and B. Bensaou, "Hysteresis-based active queue management for TCP traffic in data centers," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, 2019, pp. 1621–1629.

[7] B. Hesmans, H. Tran-Viet, R. Sadre, and O. Bonaventure, "A first look at real multipath TCP traffic," in *Proc. Int. Workshop Traffic Monitor. Anal. (TMA)*, 2015, pp. 233–246.

[8] T. Kato, A. Diwakar, R. Yamamoto, S. Ohzahata, and N. Suzuki, "Experimental analysis of MPTCP congestion control algorithms; LIA, OLIA and BALIA," in *Proc. Int. Conf. Theory Pract. Modern Comput. (TPMC)*, 2019, pp. 135–142.

[9] R. Barik, M. Welzl, S. Ferlin, and O. Alay, "LISA: A linked slow-start algorithm for MPTCP," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2016, pp. 1–7.

[10] Q. Li, M. Dong, and P. B. Godfrey, "Halfback: Running short flows quickly and safely," in *Proc. Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2015, pp. 1–13.

[11] Y. Thomas, M. Karaliopoulos, G. Xylomenos, and G. C. Polyzos, "Low latency friendliness for multipath TCP," *IEEE/ACM Trans. Netw.*, vol. 28, no. 1, pp. 248–261, Feb. 2020.

[12] M. Kheirkhah, I. Wakeman, and G. Parisis, "Short vs. long flows: A battle that both can win," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 349–350, 2015.

[13] C. Lee, J. Jung, and J.-M. Chung, "DEFT: Multipath TCP for high speed low latency communications in 5G networks," *IEEE Trans. Mobile Comput.*, vol. 20, no. 12, pp. 3311–3323, Dec. 2021.

[14] W. Li, H. Zhang, S. Gao, C. Xue, X. Wang, and S. Lu, "SmartCC: A reinforcement learning approach for multipath TCP congestion control in heterogeneous networks," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 11, pp. 2621–2633, Nov. 2019.

[15] Z. Xu, J. Tang, C. Yin, Y. Wang, and G. Xue, "Experience-driven congestion control: When multi-path TCP meets deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1325–1336, Jun. 2019.

[16] S. R. Pokhrel and A. Walid, "Learning to harness bandwidth with multipath congestion control and scheduling," *IEEE Trans. Mobile Comput.*, vol. 22, no. 2, pp. 996–1009, Feb. 2023.

[17] B. He et al., "DeepCC: Multi-agent deep reinforcement learning congestion control for multi-path TCP based on self-attention," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 4, pp. 4770–4788, Dec. 2021.

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

[19] M. Kheirkhah, I. Wakeman, and G. Parisis, "MMPTCP: A multipath transport protocol for data centers," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, 2016, pp. 1–9.

[20] P. Dong et al., "Reducing transport latency for short flows with multipath TCP," *J. Netw. Comput. Appl.*, vol. 108, pp. 20–36, Apr. 2018.

[21] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Trans. Netw.*, vol. 8, no. 5, pp. 556–567, Oct. 2000.

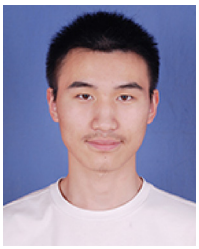
[22] Y. Wang, K. Xue, H. Yue, J. Han, Q. Xu, and P. Hong, "Coupled slow-start: Improving the efficiency and friendliness of MPTCP's slow-start," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2017, pp. 1–6.
[23] V. Jacobson. "Modified TCP congestion avoidance algorithm." 1990. [Online]. Available: http://www.postel.org/pipermail/end2end-interest/
[24] A. Gurtov, T. Henderson, S. Floyd, and Y. Nishida, "The NewReno modification to TCP's fast recovery algorithm," IETF, RFC 6582, 2012. Accessed: Jan. 2023. [Online]. Available: https://www.ietf.org/rfc/rfc6582.txt
[25] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 83–91, 2003.
[26] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM Spec. Interest Group Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.
[27] W. Sun, L. Xu, S. Elbaum, and D. Zhao, "Model-agnostic and efficient exploration of numerical congestion control state space of real-world TCP implementations," *IEEE/ACM Trans. Netw.*, vol. 29, no. 5, pp. 1990–2004, Oct. 2021.
[28] M. Chen, R. Li, J. Crowcroft, J. Wu, Z. Zhao, and H. Zhang, "RAN information-assisted TCP congestion control using deep reinforcement learning with reward redistribution," *IEEE Trans. Commun.*, vol. 70, no. 1, pp. 215–230, Jan. 2022.
[29] S. Fu and M. Atiquzzaman, "SCTP: State of the art in research, products, and technical challenges," *IEEE Commun. Mag.*, vol. 42, no. 4, pp. 64–76, Apr. 2004.
[30] M. Honda, Y. Nishida, L. Eggert, P. Sarolahti, and H. Tokuda, "Multipath congestion control for shared bottleneck," in *Proc. Int. Workshop Protocols Future Large-Scale Diverse Netw. Transports (PFLDNT)*, vol. 357, 2009, p. 378.
[31] 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.
[32] D.-G. Zhang, K. Zheng, D.-X. Zhao, X.-D. Song, and X. Wang, "Novel quick start (QS) method for optimization of TCP," *Wireless Netw.*, vol. 22, no. 1, pp. 211–222, 2016.
[33] W. Bai, S. Hu, K. Chen, K. Tan, and Y. Xiong, "One more config is enough: Saving (DC)TCP for high-speed extremely shallow-buffered datacenters," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 489–502, Apr. 2021.
[34] 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.
[35] W. Wei, K. Xue, J. Han, D. S. L. Wei, and P. Hong, "Shared bottleneck-based congestion control and packet scheduling for multipath TCP," *IEEE/ACM Trans. Netw.*, vol. 28, no. 2, pp. 653–666, Apr. 2020.
[36] M. Allman, "TCP byte counting refinements," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 29, no. 3, pp. 14–22, 1999.
[37] J. C. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 4, pp. 270–280, 1996.
[38] S. Hassayoun, J. Iyengar, and D. Ros, "Dynamic window coupling for multipath congestion control," in *Proc. IEEE Int. Conf. Netw. Protocols (ICNP)*, 2011, pp. 341–352.
[39] S. Ferlin, Ö. Alay, T. Dreibholz, D. A. Hayes, and M. Welzl, "Revisiting congestion control for multipath TCP with shared bottleneck detection," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, 2016, pp. 1–9.
[40] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and Tools for Network Simulation*. Berlin, Germany: Springer, 2010, pp. 15–34.
[41] "MPTCP NS3 code." Accessed: Jan. 2023. [Online]. Available: http://code.google.com/p/mptcp-ns3/
[42] "Apache2." Accessed: Jan. 2023. [Online]. Available: https://www.apache.org/
[43] "Multipath TCP—Linux kernel implementation." Accessed: Jan. 2023. [Online]. Available: http://www.multipath-tcp.org/
[44] "Iperf." Accessed: Jan. 2023. [Online]. Available: https://iperf.fr/
[45] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," Eastern Res. Lab., Digit. Equip. Corp., Hudson, MA, USA, Rep. DEC-TR-301, 1984. Accessed: Jan. 2023. [Online]. Available: https://ocw.cs.pub.ro/courses/_media/isrm/laboratoare/new/a_quantitative_measure_of_fairness_and_d.pdf

Jiayu Yang (Graduate Student Member, IEEE) received the B.S. degree in information security from the School of Cyber Security, University of Science and Technology of China in 2019, where she is currently pursuing the Ph.D. degree in information Security with the School of Cyber Science and Technology. Her research interests include future Internet architecture design, transmission optimization, and network security.

Jiangping Han (Member, IEEE) received the B.S. and Ph.D. degrees from the Department of Electronic Engineering and Information Science, USTC in 2016 and 2021, respectively, where she is currently a Postdoctoral Fellow with the School of Cyber Science and Technology. From November 2019 to October 2021, she was a Visiting Scholar with the School of Computing, Informatics, and Decision Systems Engineering, Arizona State University. She is currently a Postdoctoral Researcher with the School of Cyber Science and Technology, USTC. Her 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, USTC in 2007. From May 2012 to May 2013, he was a Postdoctoral Researcher with Department of Electrical and Computer Engineering, University of Florida. He is currently a Professor with the School of Cyber Science and Technology, USTC. He has authored and coauthored more than 100 technical papers in various archival journals and conference proceedings. His research interests include next generation Internet architecture design, transmission optimization, and network security. He serves on the editorial board of several journals, including the IEEE TRANSACTIONS OF DEPENDABLE AND SECURE COMPUTING, the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, and IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT. He has also served as a (Lead) Guest Editor for many reputed journals/magazines, including IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, *IEEE Communications Magazine*, and IEEE NETWORK. He is an IET Fellow.

Yansen Wang received the bachelor's degree from the Department of Communication Engineering, University of Electronic Science and Technology of China in 2015, and the master's degree from the Department of Electronic Engineering and Information Science, USTC. He is currently an Engineer with the Natural Language Processing Research Department, Meituan Group, China. His research interests include transmission optimization and natural language processing.



Jian Li (Member, IEEE) received the bachelor's degree from the Department of Electronics and Information Engineering, Anhui University in 2015, and the Ph.D. degree from the Department of Electronic Engineering and Information Science, University of Science and Technology of China (USTC) in 2020. From November 2019 to November 2020, he was a Visiting Scholar with the Department of Electronic and Computer Engineering, University of Florida. From December 2020 to October 2022, he was a Postdoctoral Researcher with the School of Cyber Science and Technology, USTC, where he is currently an Associate Researcher. His research interests include wireless networks, next-generation Internet, and quantum networks.

Yitao Xing (Graduate Student Member, IEEE) received the B.S. degree in information security from the School of the Gifted Young, University of Science and Technology of China in 2018, where he is currently pursuing the Ph.D. degree with the School of Cyber Science and Technology. His research interests include future Internet architecture and transmission optimization.

Hao Yue (Member, IEEE) received the B.Eng. degree in telecommunication engineering from Xidian University, Xi'an, China, in 2009, and the Ph.D. degree in electrical and computer engineering from the University of Florida, Gainesville, FL, USA, in 2015. He is currently an Associate Processor with the Department of Computer Science, San Francisco State University, San Francisco, CA, USA. His research interests include cyber-physical systems, cybersecurity, wireless networking, and mobile computing.

David S. L. Wei (Life 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, University of Aizu, Japan, (as an Associate Professor and then a Professor). He is currently a Professor with the Computer and Information Science Department, Fordham University. He has authored and coauthored more than 140 technical papers in various archival journals and conference proceedings. He currently focuses his research efforts on cloud and edge computing, cybersecurity, and quantum computing and communications. He is the recipient of IEEE Region 1 Technological Innovation Award (Academic) in 2020, for contributions to information security in wireless and satellite communications and cyber-physical systems. He was a Lead Guest Editor or a 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 IEEE TRANSACTIONS ON CLOUD COMPUTING from 2014 to 2018, an Editor of IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS for the Series on Network Softwarization and Enablers from 2018 to 2020, and an Associate Editor of *Journal of Circuits, Systems and Computers* from 2013 to 2018. He is a member of ACM and AAAS and a Life Senior Member of IEEE Computer Society and IEEE Communications Society.