electronics

MDPI

Article

Adaptive Decrease Window for BALIA (ADW-BALIA): Congestion Control Algorithm for Throughput Improvement in Nonshared Bottlenecks

Geon-Hwan Kim, Yeong-Jun Song, Imtiaz Mahmud and You-Ze Cho *

School of Electronic and Electrical Engineering, Kyungpook National University, Daegu 41566, Korea; kgh76@ee.knu.ac.kr (G.-H.K.); syj5385@knu.ac.kr (Y.-J.S.); imtiaz.tee@gmail.com (I.M.)
* Correspondence: yzcho@ee.knu.ac.kr

Abstract: The main design goals of the multipath transmission control protocol (MPTCP) are to improve the throughput and share a common bottleneck link fairly with a single-path transmission control protocol (TCP). The existing MPTCP congestion control algorithms achieve the goal of fairness with single-path TCP flows in a shared bottleneck, but they cannot maximize the throughput in nonshared bottlenecks, where multiple subflows traverse different bottleneck links. This is because the MPTCP is designed not to exceed the throughput of a single-path TCP competing in the bottleneck. Therefore, we believe that MPTCP congestion control should have different congestion window control mechanisms, depending on the bottleneck type. In this paper, we propose an adaptive decrease window (ADW) balanced linked adaptation (BALIA) congestion control algorithm that adaptively adjusts the congestion window decrease in order to achieve better throughput in nonshared bottlenecks while maintaining fairness with the single-path TCP flows in shared bottlenecks. The ADW-BALIA algorithm detects shared and nonshared bottlenecks based on delay fluctuations and it uses different congestion window decrease methods for the two types of bottleneck. When the delay fluctuations of the MPTCP subflows are similar, the ADW-BALIA algorithm behaves the same as the existing BALIA congestion control algorithm. If the delay fluctuations are dissimilar, then the ADW-BALIA algorithm adaptively modulates the congestion window reduction. We implement the ADW-BALIA algorithm in the Linux kernel and perform an emulation experiment that is based on various topologies. ADW-BALIA improves the aggregate MPTCP throughput by 20% in the nonshared bottleneck scenario, while maintaining fairness with the single-path TCP in the shared bottleneck scenario. Even in a triple bottleneck topology, where both types of bottlenecks exist together, the throughput increases significantly. We confirmed that the ADW-BALIA algorithm works stably for different delay paths, in competition with CUBIC flows, and with lossy links.

Keywords: multipath transmission control protocol (MPTCP); congestion control; nonshared bottleneck; fairness; throughput

Citation: Kim, G.-H.; Song, Y.-J.; Mahmud, I.; Cho, Y.-Z. Adaptive Decrease Window for BALIA (ADW-BALIA): Congestion Control Algorithm for Throughput Improvement in Nonshared Bottlenecks. *Electronics* **2021**, *10*, 294. https://doi.org/10.3390/electronics10030294

Academic Editor: Jaime Lloret Mauri
Received: 10 November 2020
Accepted: 20 January 2021
Published: 26 January 2021

1. Introduction

The transport layer protocol provides an end-to-end connection over the Internet. The most widely used transport protocol is the transmission control protocol (TCP), which is designed to offer reliable data transfer [1]. In an end-to-end connection, TCP congestion control has a direct effect on network performance and it increases or decreases the congestion window (cwnd) in order to control the sending rate at the source host [2]. TCP Reno is the most typical and basic congestion control algorithm [2,3], which uses the packet loss as an indicator of network congestion [4].

Recently, cellular networks have presented new challenges for TCP, which are related to the concurrent use of multiple interfaces, such as Wi-Fi and 4G/5G in mobile devices [5,6]. The simultaneous use of multiple interfaces of multi-homed devices can meet the increasing demand for communications [7]. However, the traditional TCP does not

support the use of multiple available interfaces and it only uses one path for an end-to-end connection [8]. Therefore, the Internet Engineering Task Force created the multipath TCP (MPTCP) working group, and MPTCP was introduced [9,10].

MPTCP is a set of regular TCP extensions for providing a multipath TCP service, allowing the transport connection to simultaneously operate on multiple paths [10]. A sequence of TCP segments transmitting on individual paths that form part of an MPTCP connection is defined as a subflow [9]. MPTCP allows for a single data stream to be sent over multiple paths [11]. Dividing the data stream by subflows between the hosts [12] can maximize resource utilization and increase the end-to-end throughput. Moreover, MPTCP provides better robustness and resilience against network/link failures and the performance degradation of some paths. Multi-homed devices using MPTCP can support continuous connections while changing the network interface.

The three design goals of MPTCP that must be met in order to ensure better performance and fairness are as follows [13]:

- Goal 1 (Throughput improvement): the total throughput of an MPTCP connection should be higher than that of a single TCP using the best path.
- Goal 2 (Fairness to TCP): when multiple subflows share a common bottleneck link with a single-path TCP flow, the aggregate throughput of an MPTCP connection should be equal to that of a single-path TCP.
- Goal 3 (Load balancing): MPTCP should prefer the best paths, and more traffic should be sent to less-congested paths.

The first goal is the reason multiple paths are used [9], whereas the second goal is essential in order to coexist with the single-path TCP in a common bottleneck link. The third goal must be considered in order to improve the performance of MPTCP after achieving the first two goals [14]. Therefore, Goals 1 and 2 must be achieved first, primarily by the MPTCP congestion control algorithm that allocates network resources between flows passing through the link. MPTCP congestion control, which adjusts the congestion window of several subflows, plays a more important role than traditional TCP congestion control, because MPTCP is an extension of TCP.

In order to achieve these goals, additive increase/multiplicative decrease (AIMD)-based [15,16] MPTCP congestion control algorithms, such as coupled [17], linked increase adaptation (LIA) [18], optimized linked increase adaptation (OLIA) [11], and balanced linked adaptation (BALIA) [19], were proposed. The coupled congestion control algorithm focused on TCP friendliness [20], whereas the subsequent algorithms improved responsiveness and load balancing. The MPTCP congestion control algorithm that is currently implemented on Linux was designed to achieve fairness with TCP, which means that the aggregate throughput of the MPTCP connection should not be higher than that of the single-path TCP passing through the best path. While this fairness goal ensures that all of the connections take up the same bandwidth, it is unfair for the MPTCP connection.

The bottleneck link determines the end-to-end throughput of a TCP session. Because the traditional TCP uses a single path, only one bottleneck link exists. However, the aggregate throughput of an MPTCP connection is not always determined by just one bottleneck link. When all of the subflows pass through different links, there can be as many bottleneck links as the number of subflows. Nevertheless, the BALIA congestion control algorithm (and the LIA and OLIA algorithms) limits the throughput of each subflow when it passes through different bottleneck links, while assuming that the subflows pass through the same bottleneck link [21].

In this study, we propose a modified BALIA congestion control algorithm that can minimize the throughput degradation that occurs due to the fairness goal of MPTCP. We call this algorithm the adaptive decrease window (ADW) for BALIA congestion control algorithm, which adjusts the amount of congestion window reduction when congestion occurs. It continuously tracks the delay fluctuations of each subflow and, if the delay fluctuations are similar, then ADW-BALIA behaves the same as the existing BALIA algorithm to coexist fairly with the single-path TCP. If the delay fluctuations are dissimilar, the ADW-BALIA

algorithm improves the throughput by adapting the congestion window reduction, so that surplus resources that are available at nonshared links are not left unused.

The main contribution of the ADW-BALIA algorithm are as follows:

- The ADW-BALIA algorithm improves the aggregate MPTCP throughput by approximately 20% when traversing nonshared bottleneck links. This algorithm also maintains fairness with the TCP congestion control algorithm on the shared bottleneck link.
- In designing the ADW-BALIA algorithm, a major modification of the BALIA algorithm is the congestion window decrease mechanism. ADW-BALIA adaptively adjusts the congestion window reduction based on the round-trip time (RTT) fluctuation of the subflow.
- The ADW method designed in this paper was applied to the BALIA algorithm to implement the ADW-BALIA algorithm. This method can also be applied to both the LIA and OLIA algorithms.

The remainder of the paper is organized, as follows. In Section 2, we investigate the existing MPTCP congestion control algorithms and present a bottleneck type of MPTCP. We introduce the details of the ADW-BALIA congestion control algorithm in Section 3. Subsequently, we evaluate the ADW-BALIA algorithm with various topologies by conducting an emulation experiment in Section 4. Finally, Section 5 concludes the paper.

2. Related Work

2.1. Multipath Transmission Control Protocol Congestion Control Algorithm

TCP Reno was the first congestion control algorithm to be broadly deployed, but today the default algorithm in both Linux and Windows 10 operating systems is TCP CUBIC [22]. With the gradual development of transmission technology and network devices, the transmission capacity of the network significantly increased; however, the TCP congestion control algorithm did not fully use the capacity. Therefore, a new congestion control algorithm, called bottleneck bandwidth round-trip propagation time (BBR) [23], was recently proposed, which attempts to address the need for an optimal TCP congestion control algorithm that is suitable for the current network characteristics.

Similarly, a new congestion control algorithm that is suitable for MPTCP is required to use MPTCP with multiple subflows efficiently. If every subflow of an MPTCP connection operated as an independent TCP Reno connection, then the TCP fairness goal of MPTCP would be violated in the shared bottleneck case. Thus, MPTCP congestion control algorithms, such as EWTCP [24], LIA [18], OLIA [11], BALIA [19], and wVegas [25], have been proposed to use multiple subflows effectively while being fair to the traditional TCP. Most of these algorithms are based on TCP Reno and they retain the same decrease mechanism as TCP Reno, only modifying the increase mechanism of the congestion window.

The default congestion control algorithm in the Linux MPTCP, LIA, adjusts the `cwnd` on multiple routes, as follows:

- For each ACK received on subflow r, $w_r = w_r + \min(\frac{\alpha}{w_{total}}, \frac{1}{w_r})$,
- For each packet loss on subflow r, $w_r = w_r - \frac{w_r}{2}$,

where w_r is the cwnd of subflow r, α is a multipath aggressiveness factor, and w_{total} is the sum of w_r for all subflows. Therefore, the subflow of the default MPTCP connection uses not only its own cwnd, w_r, but also the cwnd of other subflows to adjust the cwnd. The modification of the increase mechanism causes the increase rate of the aggregate cwnd of all subflows to be equal to that of the single-path TCP, thus ensuring fairness to the single-path TCP. However, the decrease mechanism is retained as for TCP Reno.

Moreover, the strategy of increasing and decreasing the cwnd of the BALIA algorithm, which is the basis of the proposed algorithm in this paper, is as follows:

- For each ACK received on subflow r, $w_r = w_r + [\frac{x_r}{\tau_r x_k} \cdot (\frac{1+\alpha_r}{2}) \cdot (\frac{4+\alpha_r}{5})]$,
- For each packet loss on subflow r, $w_r = w_r - [\frac{x_r}{2} \cdot \min(\alpha_r, \frac{3}{2})]$,

where α_r is a multipath aggressiveness factor of subflow r, and τ_r is the round-trip time that is observed on r. Each variable is defined, as follows: $\alpha_r = max\{x\}/x_r$, where $x_r = w_r/\tau_r$, and $x = |\sum_{k\in\mathcal{P}}(w_k/\tau_k)|^2$. The BALIA algorithm allows for w_r oscillation up to an ideal level to provide a good balance between TCP friendliness and responsiveness. When the subflow r passes through the best path, or the MPTCP connection uses a single path, α_r is set to 1, which makes both the increase and decrease of w_r reduce to the same values with TCP Reno [26].

The fairness goal of MPTCP can also be described by the fluid model for MPTCP [19]. Equations (1) and (2) describe the fluid model, as follows:

$$\dot{x}_r = k_r(\mathbf{x}_s)(\phi_r(\mathbf{x}_s) - q_r/2)^+_{x_r} \quad (1)$$

$$\dot{p}_l = \gamma_l(y_l - c_l)^+_{p_l} \quad (2)$$

where the MPTCP session source s associates a set of paths, and each path $r \in s$ has the sending rate x_r. In addition, q_r represents the approximate packet loss probability on path r. Moreover, each link $l \in r$ has the capacity c_l, loss probability p_l, and aggregate traffic rate y_l.

The authors also provide the following TCP friendliness design condition for the gain γ_l in the MPTCP fluid model.

- Condition of fairness with TCP: for any path r of MPTCP session source s, the function $\phi_r(\mathbf{x}_s)$ in Equation (1) satisfies $\phi_r(\mathbf{x}_s) \leq (x_r\tau_r)^{-2}$.

This condition ensures that the cwnd of the MPTCP subflow increases no more aggressively than that of a single-path TCP flow [8,19].

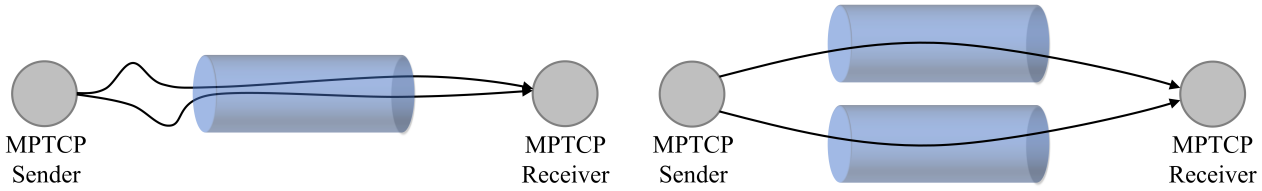
A new MPTCP congestion control has recently been proposed, which deviates from the traditional AIMD-based TCP Reno mechanism [27–31]. Several studies have been conducted as a typical example, which applied the BBR congestion control to MPTCP [28–31]. However, numerous problems have been encountered in the early version of the BBR congestion control, and the proposed multipath BBR congestion control based on the early version inherits related problems and may cause additional issues [32–34]. In addition, BBRv2 is being developed for solving the inherent problem of TCP BBR, and the multipath BBR must be entirely revised according to the change in TCP BBR.

2.2. Bottleneck Links in Multipath Transmission Control Protocol

MPTCP can have different bottleneck types, depending on the number of subflows. As depicted in Figure 1, MPTCP subflows can pass through either a different link (Figure 1b) or the same link (Figure 1a). The case in which two subflows pass through the same bottleneck link is a shared bottleneck, and the other case is a nonshared bottleneck [35]. When the MPTCP subflows compete with single-path TCP flows on a shared bottleneck link, MPTCP must meet its fairness goal to coexist with the traditional TCP. If this goal is not achieved, single-path TCP hosts cannot occupy adequate bandwidth, which results in fairness issues. Therefore, the MPTCP congestion control algorithms currently implemented in Linux [36] demonstrate sufficient fairness with the single-path TCP. However, the efforts of MPTCP to maintain fairness for the traditional TCP may cause unintended performance degradation [19]. Thus, the MPTCP throughput may be limited on a nonshared bottleneck, due to the fairness goal of MPTCP. When each subflow passes through a nonshared bottleneck link, the throughput can be maximized by not considering the fairness goal of MPTCP. However, the existing MPTCP congestion control algorithm in Linux does not achieve the maximum throughput in a nonshared bottleneck.

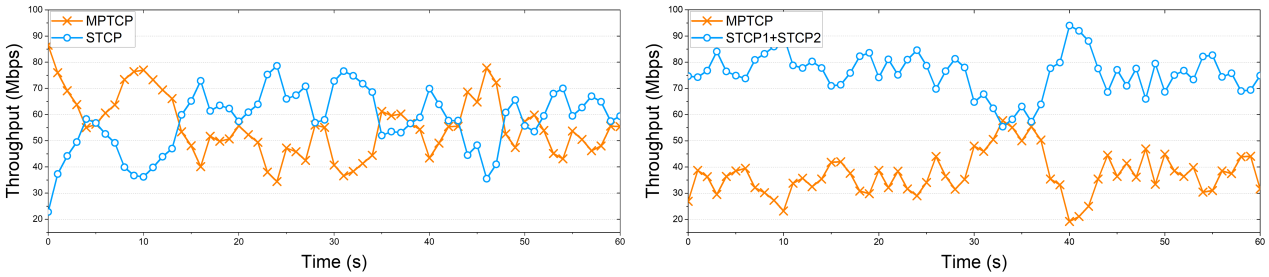
Two experiments were conducted to observe the throughput of MPTCP, according to the bottleneck type. Section 4 describes the detailed experimental environment and setup. Figure 2a illustrates the throughput of MPTCP and TCP when all flows share the same bottleneck link set to 120 Mbps in bandwidth. One single-path TCP flow competes with the MPTCP connection with two subflows; thus, the fair share is 60 Mbps. Fluctuations exist between MPTCP and TCP, but, on average, they compete fairly. In the nonshared

bottleneck scenario, each subflow of MPTCP competes with a single-path TCP flow at each bottleneck link set to a 60 Mbps bandwidth. In Figure 2b, we present the sum of the throughput of two single-path TCP flows and the aggregate MPTCP throughput. The maximum aggregate throughput that MPTCP can achieve is 60 Mbps, but it averages a throughput of 40 Mbps, which is approximately 30% lower.



(a) Shared bottleneck link (b) Nonshared bottleneck link

Figure 1. Two types of bottlenecks in the multipath transmission control protocol with two subflows.



(a) Shared bottleneck link (b) Nonshared bottleneck link

Figure 2. Throughput comparison by bottleneck type.

2.3. Shared Bottleneck Detection Algorithm

Because TCP is an end-to-end transport protocol, it cannot explicitly determine which link it passes through or the efficiency of the path. Therefore, the packet loss and delay information can typically be used to detect which flow shares the bottleneck. To detect a shared bottleneck, several studies have used various types of network information [21,22,37–41]. The first algorithm that was proposed to determine whether MPTCP shares a common bottleneck link is the dynamic window coupling algorithm by Hassayoun et al. [22]. It detects the shared bottleneck while using the increase in the delay or packet loss over a certain period. In addition, Ferlin et al. [21] proposed the shared bottleneck detection algorithm for MPTCP based on three key statistics of a one-way delay: skewness, variability, and key frequency. Unlike the above papers using delay information, Wei et al. [41] proposed a congestion control algorithm and packet scheduler for MPTCP based on the bottleneck detection mechanism using explicit congestion notification [42].

These shared bottleneck detection studies exhibited relatively high detection accuracy. However, in the network, multiple bottlenecks can occur, and shared and nonshared bottlenecks may coexist. Therefore, in this study, we provide a customized solution for the BALIA algorithm that adapts and operates on changes of bottleneck, rather than simply categorizing the bottleneck as shared or nonshared. The proposed algorithm does not harm the single-path TCP on a shared bottleneck link. In addition, it improves the aggregate MPTCP throughput on a nonshared bottleneck. Moreover, the ADW-BALIA algorithm exhibits sufficient throughput improvement in complex scenarios, in which shared and nonshared bottlenecks are combined.

3. Adaptive Decrease Window for BALIA Congestion Control Algorithm

In this section, we introduce our proposed congestion control algorithm to improve the MPTCP throughput without harming the traditional TCP flows. First, we examine

the characteristics of the delay fluctuation, according to the type of bottleneck through which the MPTCP connection passes. Subsequently, based on the difference in the delay fluctuation, we estimate the similarity between the subflows. The proposed ADW-BALIA congestion control algorithm adaptively adjusts the cwnd decrease mechanism while using the estimated similarity.

3.1. Similarity of the Round-Trip-Time Fluctuations between Subflows

The TCP throughput is determined by the bandwidth of the bottleneck link and the RTT between the end-to-end hosts. In order to achieve the maximum throughput, the available capacity of the link must be used as much as possible, and the maximum amount of data that can be transferred during one RTT period, the bandwidth-delay product (BDP), is calculated as the product of the bottleneck bandwidth and round-trip propagation time. A queue is created on multiple links along the data path of a TCP flow when the amount of inflight data exceeds the BDP. The existing loss-based congestion control algorithm continues to transmit until the bottleneck buffer is full, and the congestion control reduces the amount of transmission when a buffer overflow occurs, which causes end-to-end RTT fluctuations. The bottleneck link capacity can be estimated from the RTT fluctuations in a stable network with no random packet loss.

If the network resources of the two paths are different, the RTT fluctuations of the flows passing through each path are also different. We confirmed the pattern of RTT fluctuations through actual emulation experiments while applying this assumption to the MPTCP subflow. The experiments were configured, so that MPTCP with two subflows passes through two types of bottleneck links. Figure 3 presents the smoothed RTT of each subflow for the shared and nonshared bottlenecks that are presented in Figure 1. In both scenarios, a single-path TCP flow passes through the bottleneck link and, in the nonshared bottleneck scenario, both subflows have the same RTT.

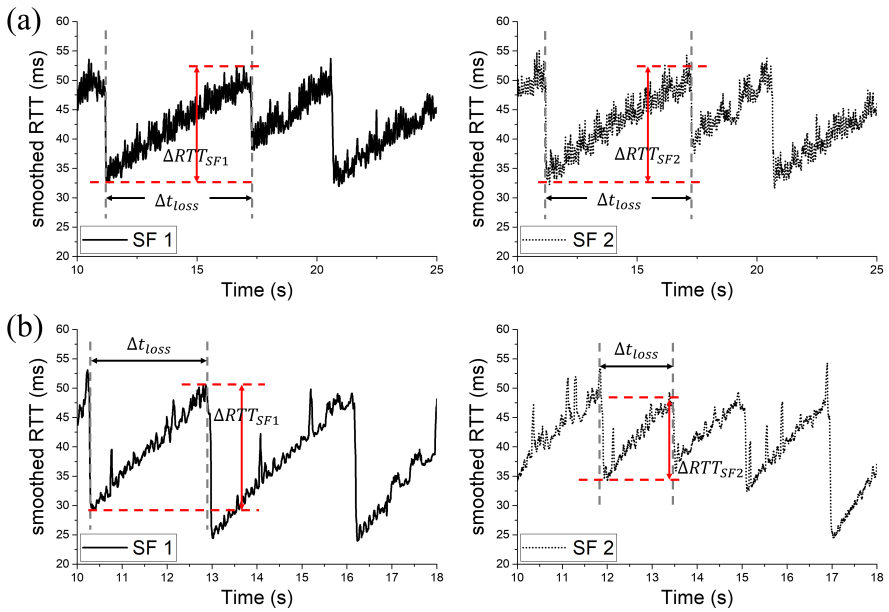


Figure 3. Round-trip-time (RTT) fluctuations according to the types of bottleneck: (**a**) both subflows coexist in the same bottleneck; (**b**) the two subflows pass through disjointed bottlenecks.

When two subflows compete on a shared bottleneck link, the RTT fluctuations between the two subflows are similar, because they fill the same bottleneck buffer. Subflows 1 (*SF1*) and 2 (*SF2*) exhibit a similar increase and decrease at 10 to 25 s, as presented in Figure 3a. The RTT difference in *SF1* between the previous packet loss event and recent packet loss event is calculated while using the following equation:

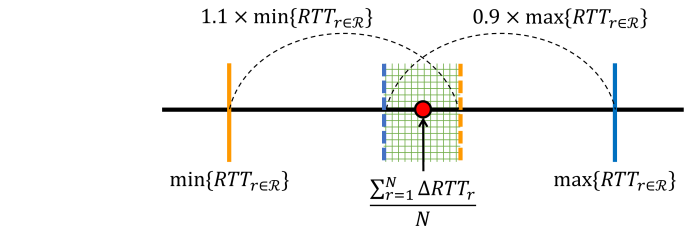
$$\Delta RTT_{SF1} = RTT_{SF1,max} - RTT_{SF1,min}. \quad (3)$$

ΔRTT_{SF1} is similar to that of *SF2*. In addition, the subsequent RTT fluctuation also coincides with the pattern of *SF2*. Instead, if two subflows pass through separate bottleneck links, the RTT fluctuations are distinct, due to their different network resources. Although the bandwidth of each bottleneck link and the RTT of both subflows were the same, ΔRTT_{SF1} and ΔRTT_{SF2} were slightly different (Figure 3b). However, the difference in the RTT fluctuations according to the scenario that is presented in Figure 3 is not completely reliable information. Even if the environment of each link is completely different, then the same RTT pattern may temporarily occur. Moreover, the RTT fluctuation can be different, even in the same environment. Therefore, the modified BALIA congestion control algorithm proposed in this paper judges that a shared bottleneck link is passed through when the RTT fluctuation similarity for each subflow is high. If the similarity is low, the cwnd decrease phase is adaptively adjusted.

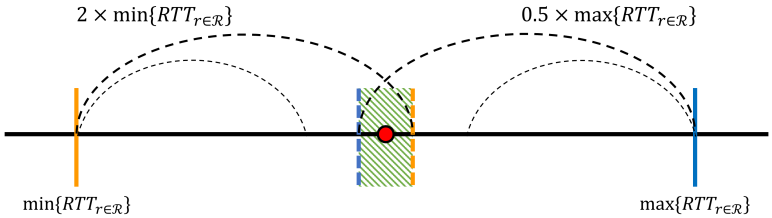
The total average value of ΔRTT_r is calculated to observe the similarity of the fluctuations for each subflow. The minimum and maximum RTT are tracked by comparing the current measured RTT to the previous RTT each time that an ACK is received. The measured minimum and maximum RTT are initialized after performing the ADW algorithm when congestion occurs due to packet loss. The RTT difference between the previous packet loss event and the most recent packet loss event is calculated. All of these variables are calculated separately for each subflow. The smoothed total average value is computed while using the exponential weighted moving average (EWMA) of the running total average sample:

$$Average_s(i) = (1 - \beta) \times Average_s(i-1) + \beta \times Average(i), \quad (4)$$

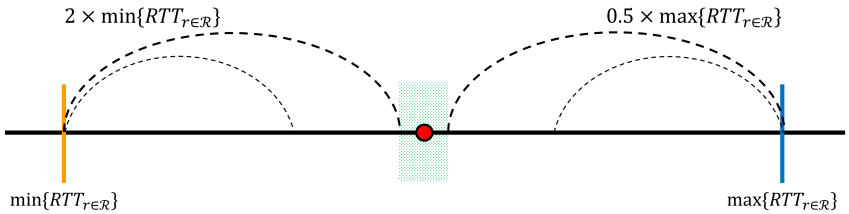
where β is a weighting factor set to $5/8$. $Average(i)$ is the latest calculated total average sample. This value is always located between $max\{RTT_{r \in R}\}$ and $min\{RTT_{r \in R}\}$, denoting the maximum and minimum RTT of the subflow r of the MPTCP connection R. Figure 4 presents three regions for the adaptive `cwnd` decrease phase. If the average value that is calculated using Equation (4) is located within the 10% boundary of the highest RTT_{max} and lowest RTT_{min} of the subflows, it is determined that the RTT fluctuations of all subflows are similar. If the $Average_s(i)$ value exceeds the 10% boundary and is located within twice the range, the `cwnd` decrease is reduced when compared to the original `cwnd` decrease, as presented in Figure 4b. Figure 4c presents the case in which the $Average_s(i)$ value is not within twice the range. In this case, the similarity of the RTT fluctuation between subflows is judged to be significantly different; thus, the `cwnd` decrease is further reduced compared with that presented in Figure 4b. The introduced boundaries were established based on experiments in shared and nonshared environments and they were tuned to the values that maintain TCP fairness in shared bottlenecks and lead to the best performance improvements in nonshared bottlenecks.



(a) Adaptive decrease region 1



(b) Adaptive decrease region 2



(c) Adaptive decrease region 3

Figure 4. Three regions for the adaptive window decrease.

3.2. Adaptive Decrease Window for Throughput Improvement in a Nonshared Bottleneck

We proposed a congestion control algorithm that adaptively selects the `cwnd` decrease for the packet loss event to improve the aggregate MPTCP throughput in a nonshared bottleneck link while maintaining fairness to the single-path TCP in a shared bottleneck link. Before designing the new algorithm, we evaluated which MPTCP congestion control algorithm exhibits the best fairness to the traditional TCP. Among them, the BALIA algorithm demonstrated better throughput for both bottleneck types. Therefore, we implemented the ADW algorithm by modifying the BALIA congestion control algorithm. However, we found that not all of the existing MPTCP algorithms have ideal throughput and coexist with a single-path TCP in a shared bottleneck link. Therefore, rather than focusing on detecting bottleneck types very accurately, the ADW algorithm aims to enable MPTCP to have the best throughput while competing fairly with the single-path TCP.

The BALIA algorithm balances the tradeoff between responsiveness and TCP friendliness of MPTCP. Peng et al. [19], who proposed BALIA, mentioned that no best parameter setting exists, because there is a tradeoff between all of the performance metrics. After identifying the design criteria for MPTCP algorithm, they experimentally tuned and chose parameters that have a good balance between responsiveness, TCP friendliness, and window oscillation. However, because the BALIA algorithm is also designed in consideration of TCP friendliness on a shared bottleneck link, it does not exhibit sufficient performance in a nonshared bottleneck. Therefore, in order to satisfy TCP friendliness, the cwnd increase phase was left as is, and only the decrease phase was changed. The cwnd increase phase of the ADW-BALIA algorithm is the same as the BALIA algorithm to be fair for the single-path TCP on a shared bottleneck. However, in the decrease phase, the reduction of the cwnd can be smaller than the original BALIA algorithm, and the ADW-BALIA algorithm can occupy the surplus resources of the link, inducing improved throughput on a nonshared bottleneck link.

If the RTT fluctuations of each subflow are similar, then each subflow is judged to be passing through the same bottleneck link and it behaves the same as in the cwnd decrease phase of the existing BALIA algorithm. Conversely, if the RTT fluctuations are not similar, then it is assumed that the subflow does not traverse the shared bottleneck link, and the cwnd reduction is attenuated to improve the aggregate throughput. This improvement comes from compensating for link underutilization, due to the excessive cwnd reduction upon congestion, which is a disadvantage of the AIMD-based congestion control. In Figure 5, the flowchart presents a detailed operation of the ADW-BALIA congestion control algorithm, and the Compensation part is a major modification to improve the throughput.

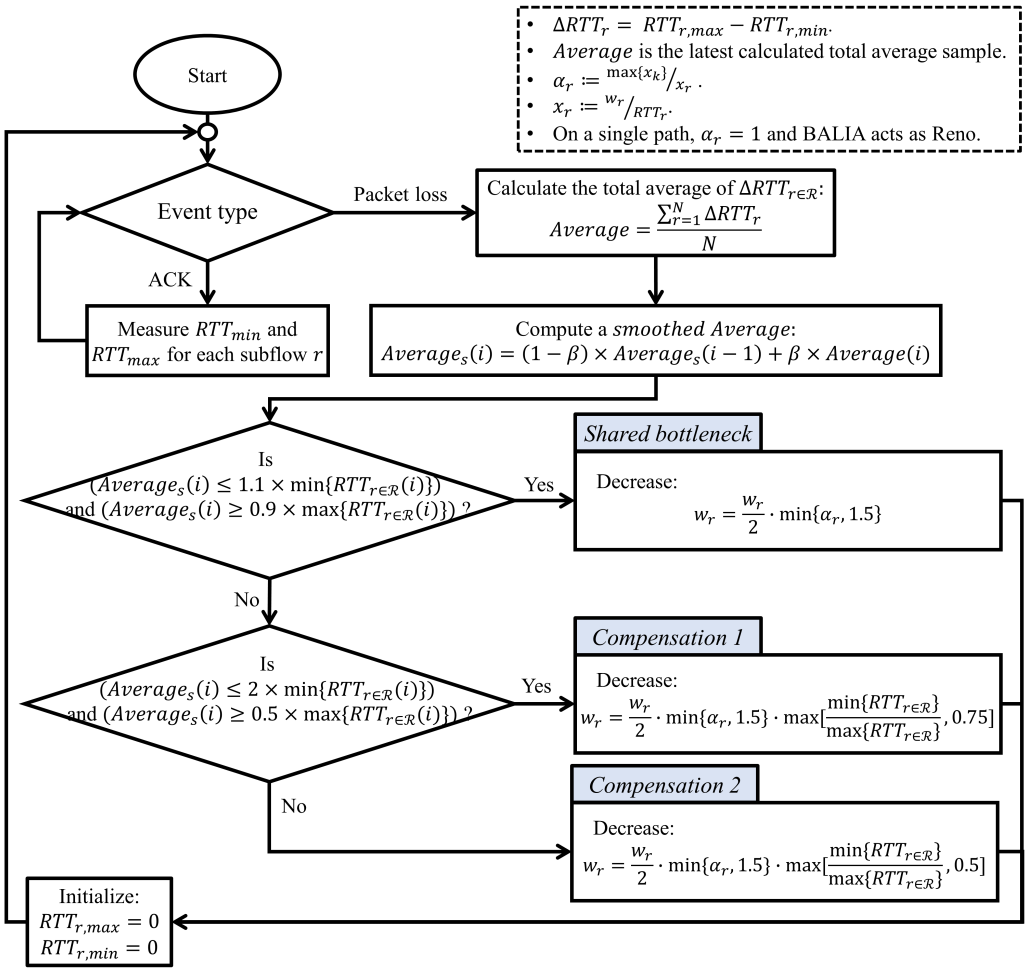


Figure 5. Flowchart of the adaptive decrease window (ADW) congestion control algorithm.

The BALIA algorithm reduces the cwnd to a minimum when 1.5 is selected in the $min()$ function, which is $1/4$ less than the previous cwnd. Based on this value, Compensation 1 sets a decreasing constant of 0.75, which is further reduced by $1/4$ from the minimum reduction of the BALIA algorithm. The $max()$ function is defined to ensure the stable operation of the ADW-BALIA algorithm in environments with a small bottleneck buffer. In the same way, Compensation 2 is further reduced by an additional $1/4$ in order to obtain more surplus resources.

As the proposed ADW-BALIA uses the RTT fluctuation, the operation of algorithm mainly depends on the queueing delay at the bottleneck buffer. Moreover, ADW-BALIA is designed by assuming the tail-drop queue management as the active queue management (AQM) algorithm at the buffer of the routers. There is a possibility that the application of different AQM algorithms might affect ADW-BALIA's operation. Therefore, the effect of

various AQM algorithms on the MPTCP congestion control algorithm will be investigated and ADW-BALIA will be upgraded further in future work.

4. Experiments and Evaluations

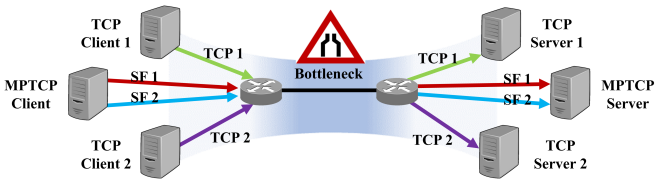
In this section, the performance of the ADW-BALIA congestion control algorithm is evaluated. We conducted an emulation experiment that is based on the Mininet emulator [43]. The Mininet environment was constructed on Ubuntu using the Linux Kernel v. 4.04 with MPTCP v. 0.90.3. For the setting of the detailed experiment, Netem [44] was used to configure the RTT, bandwidth, loss rate, and bottleneck buffer size for each link. Each client host used Iperf3 [45] to send data to the MPTCP server host. The bottleneck buffer size in all scenarios was set to 1 BDP in order to lead to periodic congestion events.

4.1. Topologies for the Experiment

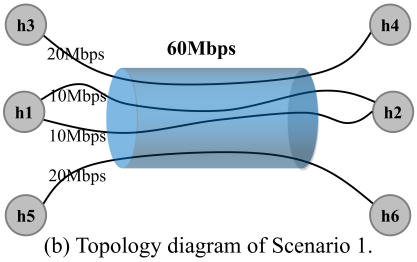
Various topologies, as presented in Figure 6, were constructed to evaluate the fairness for TCP Reno while using the existing MPTCP congestion control algorithms and the proposed ADW-BALIA algorithms and to compare the throughput in nonshared bottlenecks. First, Scenario 1 is an experiment to check whether the second design goal of MPTCP is satisfied. The MPTCP connection with two subflows shares a bottleneck link with two single-path TCP flows (Figure 6a). In Scenario 1, the throughput of one single-path TCP and the aggregate MPTCP throughput should be the same, so that the fairness goal of the MPTCP can be achieved. Figure 6b presents the ideal throughput that each flow should have in this experiment, and it illustrates a simplified topology. In addition, Scenario 2 in Figure 6c was constructed in order to check the performance on a nonshared bottleneck link, where the existing MPTCP congestion control algorithms do not achieve the ideal throughput. Likewise, Scenario 2 is simply presented again in Figure 6d, it and is ideal when all subflows have the same throughput.

The previous two topologies are the most common experimental scenarios for the performance evaluation of the MPTCP congestion control algorithm. Thus far, in the MPTCP congestion control evaluation experiments, each subflow has been limited to two cases, either sharing a bottleneck link or not. For the experiment in more diverse topologies, Scenarios 3, 4, and 5 were constructed by combining the two previous scenarios. Scenarios 3, 4, and 5 are not expected to significantly differ from the previous experiments, because the end-to-end throughput of the TCP is determined by the most congested bottleneck links in the network. However, the throughput in the situation of passing through consecutive bottleneck links was not the same as that in a simple scenario. Therefore, we evaluated the performance of the MPTCP congestion control algorithm in an environment combining two simple topologies.

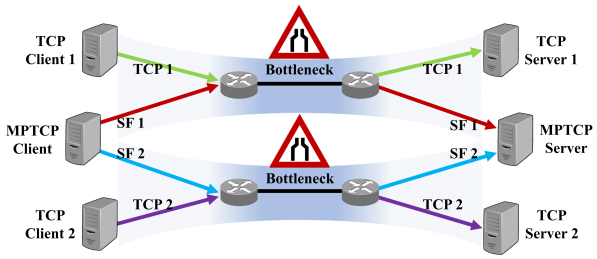
Figure 6e presents a scenario in which MPTCP competes with a single-path TCP flow in a nonshared link at first, but then competes all together in a subsequent shared bottleneck link. In other words, it is an experiment in which a larger load is applied to the shared bottleneck link in the downstream (i.e., the left side of the shared bottleneck in Figure 6e. All of the bottleneck links have a bandwidth of 60 Mbps; however, the shared bottleneck link in the downstream is more congested due to the competition of four flows. Figure 6g presents the same topology as before, but the nonshared bottleneck link is more congested, and it is expected to yield results similar to Scenario 2, because the bottleneck link is upstream. The last scenario, Figure 6i, depicts a situation where two types of bottleneck links exist simultaneously. Table 1 presents the setting values of the network parameters that were used in the experiment.



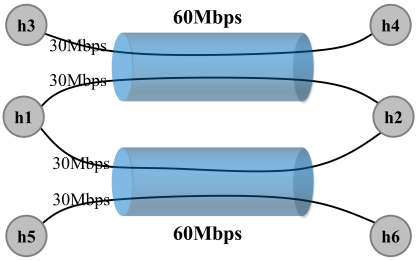
(a) Scenario 1: Shared bottleneck.



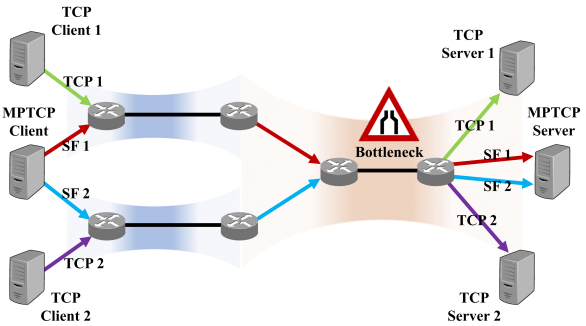
(b) Topology diagram of Scenario 1.



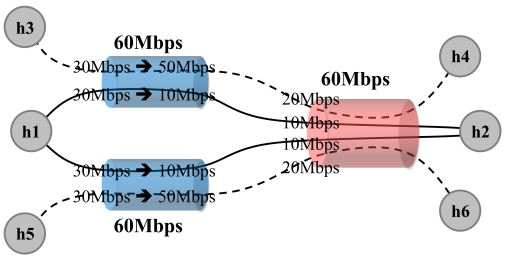
(c) Scenario 2: Nonshared bottleneck.



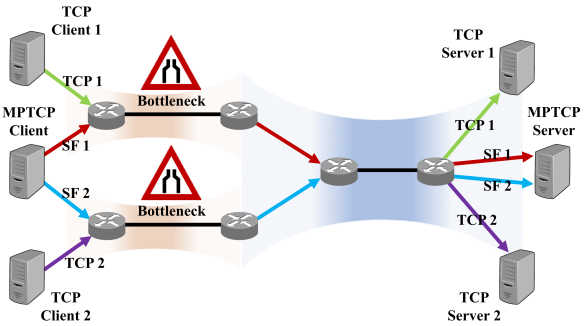
(d) Topology diagram of Scenario 2.



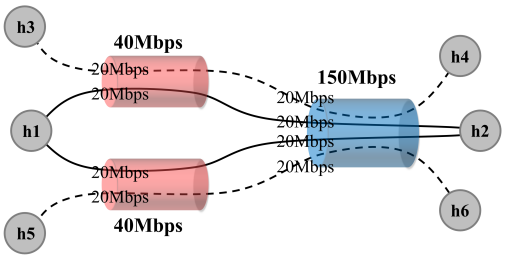
(e) Scenario 3: Combined but more congested in a shared bottleneck.



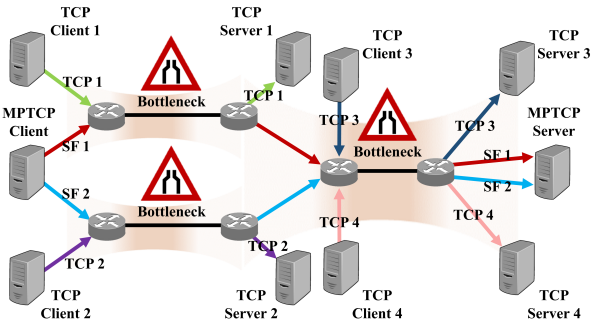
(f) Topology diagram of Scenario 3.



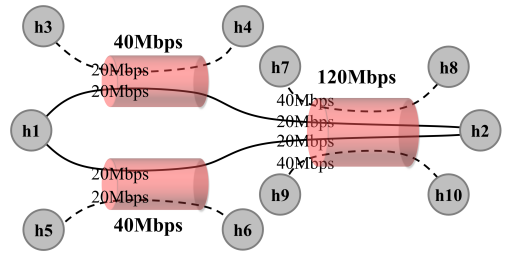
(g) Scenario 4: Combined but more congested in a nonshared bottleneck.



(h) Topology diagram of Scenario 4.



(i) Scenario 5: Triple bottleneck.



(j) Topology diagram of Scenario 5.

Figure 6. Experiment topologies and simple diagrams.

Table 1. Experimental parameters for each scenario.

	RTT	Bandwidth	Bottleneck Buffer Size	Bottleneck Bandwidth	Ideal Aggregate Throughput of MPTCP	Ideal Throughput of Single-Path TCP
Scenario 1	10 ms	100 Mbps	1 BDP	60 Mbps	20 Mbps	20 Mbps for each
Scenario 2				60 Mbps for each	60 Mbps	30 Mbps for each
Scenario 3	20 ms	100 Mbps		60 Mbps for each	20 Mbps	20 Mbps for each
Scenario 4		150 Mbps		40 Mbps for NSB 150 Mbps for SB		
Scenario 5	10 ms	120 Mbps		40 Mbps for NSB 120 Mbps for SB	40 Mbps	20 Mbps for h3, h5 40 Mbps for h7, h9

Notes: BDP: bandwidth-delay product; RTT: round-trip time; TCP: transmission control protocol; MPTCP: multipath TCP; NSB: nonshared bottleneck; SB: shared bottleneck.

4.2. Experimental Results

4.2.1. Comparison of the Throughput Ratio

Figure 7a–f present the throughput ratio for a fair bandwidth when the proposed ADW-BALIA and existing MPTCP congestion control algorithms compete with a single-path TCP in a shared bottleneck link. Although all of the sexisting MPTCP congestion control algorithms do not harm the single-path TCP, they are located between 0.8 and 0.9 and have a slightly lower throughput ratio (Figure 7a–c). However, the ADW-BALIA that is proposed by modifying the BALIA algorithm occupies more bandwidth and exhibits higher throughput when compared to the other algorithms. This is a false negative of the ADW algorithm's misjudgment, and it has no negative influence and increases the throughput in a shared bottleneck. Moreover, the two single-path TCP flows are close to their fair share when competing with the ADW-BALIA flows; thus, the fairness goal of MPTCP is sufficiently guaranteed. Figure 7e shows the results of three single-path TCP flows in a shared bottleneck with large deviations, but distributed near the fair number of 1.0. Here, the independent Reno represents MPTCP capable devices with the MPTCP option disabled, i.e., the MPTCP client host operates as a single path TCP and only uses one path to transfer data. We designed the independent Reno experiment in order to observe how the MPTCP capable communicating devices perform in the shared bottleneck when they use only one of their paths. However, we only performed this experiment in Scenario 1 because it has no significance in the nonshared bottleneck. Figure 7f illustrates that the aggregate MPTCP throughput ratio 1.5 times higher than the ideal ratio, taking up to twice as much bandwidth as each single TCP flow, results in an unfair outcome when two MPTCP subflows operate as the uncoupled Reno.

If each MPTCP subflow passes through a different path, it is ideal to occupy the same bandwidth as the single-path TCP flow at each link. Figure 8a–e present the throughput ratio for each flow divided by the maximum throughput when each subflow coexists with a single-path TCP flow in a nonshared bottleneck link. When MPTCP operates as the LIA algorithm, the throughput is about 25% lower than the maximum throughput due to the competing single-path TCP flow, as presented in Figure 8a. Figure 8b presents a significantly large deviation for OLIA and, in the worst case, nearly half of the fair throughput ratio is achieved. The BALIA algorithm has the best performance among the existing MPTCP congestion control algorithms. The proposed ADW-BALIA algorithm exhibits more improved fairness than the BALIA algorithm by mitigating the cwnd reduction based on the dissimilarity of the RTT fluctuations that may appear in the nonshared bottleneck link. Figure 8d illustrates that the first subflow exhibits some variation in the throughput ratio when compared to the second flow. However, the aggregate throughput of the ADW-BALIA algorithm is almost fair. In the case of the uncoupled congestion

control algorithm operating both subflows as the general TCP Reno, MPTCP occupies approximately 1.2 times more bandwidth than the single-path TCP.

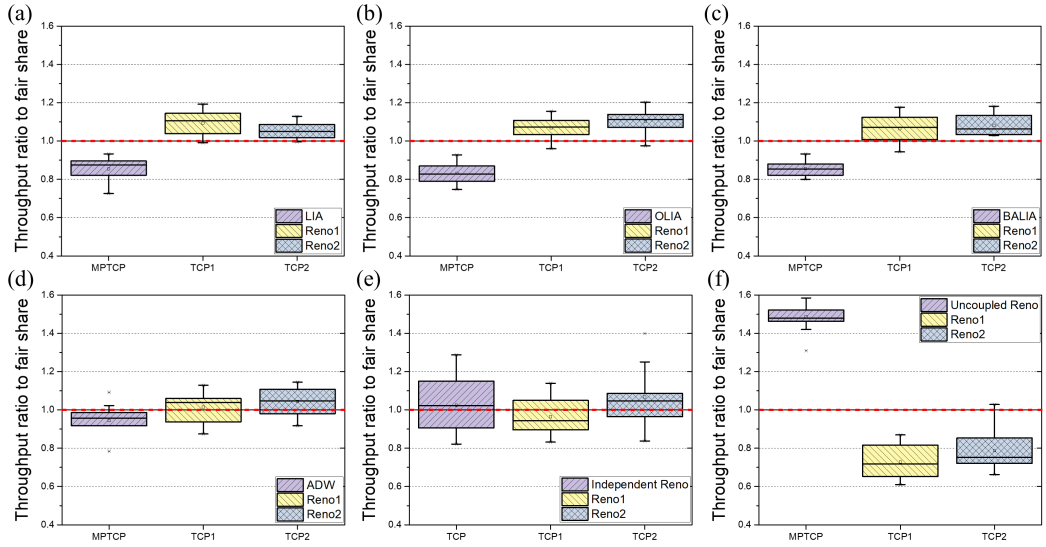


Figure 7. Evaluation results of Scenario 1: (**a**) linked increase adaptation (LIA); (**b**) optimized LIA (OLIA); (**c**) balanced LIA (BALIA); (**d**) adaptive decrease window (ADW); (**e**) independent Reno; and (**f**) uncoupled Reno.

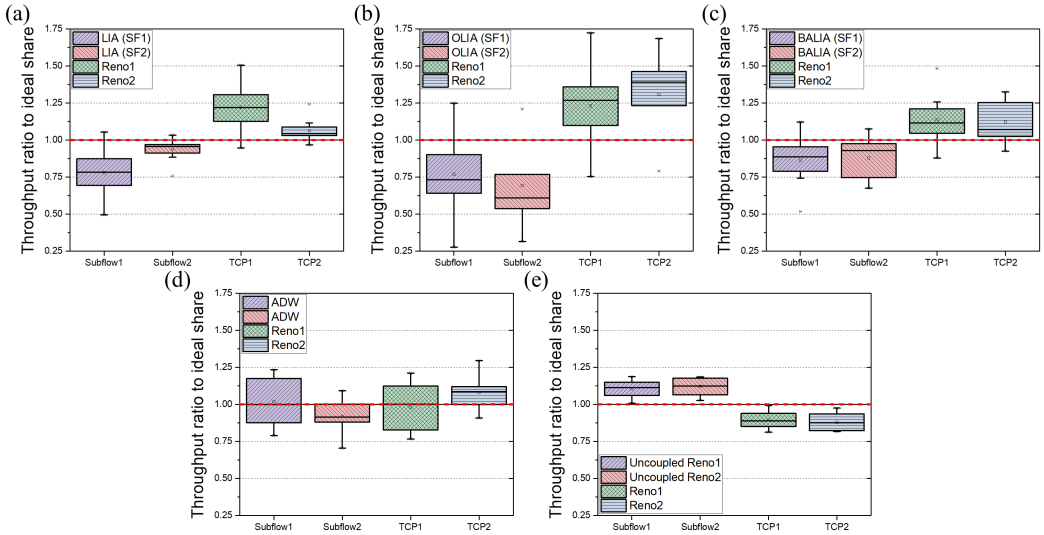


Figure 8. Evaluation results of Scenario 2: (**a**) linked increase adaptation (LIA); (**b**) optimized LIA (OLIA); (**c**) balanced LIA (BALIA); (**d**) adaptive decrease window (ADW); and, (**e**) uncoupled Reno.

Scenario 3 considers a situation where three links coexist; however, a larger load is applied to the shared bottleneck link, which is located in the downstream. The shared bottleneck link at the downstream becomes the main bottleneck, as determining the end-to-end throughput in the network is one bottleneck link, which is most congested.

Overall, the throughput ratio is reduced as compared to Scenario 1, where only one shared bottleneck exists (Figure 9a–c). When compared to other MPTCP congestion control algorithms with throughput ratios of 0.7 to 0.8, the ADW-BALIA algorithm is located between 0.8 and 0.9, showing improved throughput. The uncoupled Reno occupies considerable bandwidth, which results in unfairness with the single-path TCP because the MPTCP fairness goal is not considered when MPTCP operates as an uncoupled congestion control algorithm. Therefore, an MPTCP that does not consider the fairness to the single-path TCP can cause serious throughput degradation to TCP.

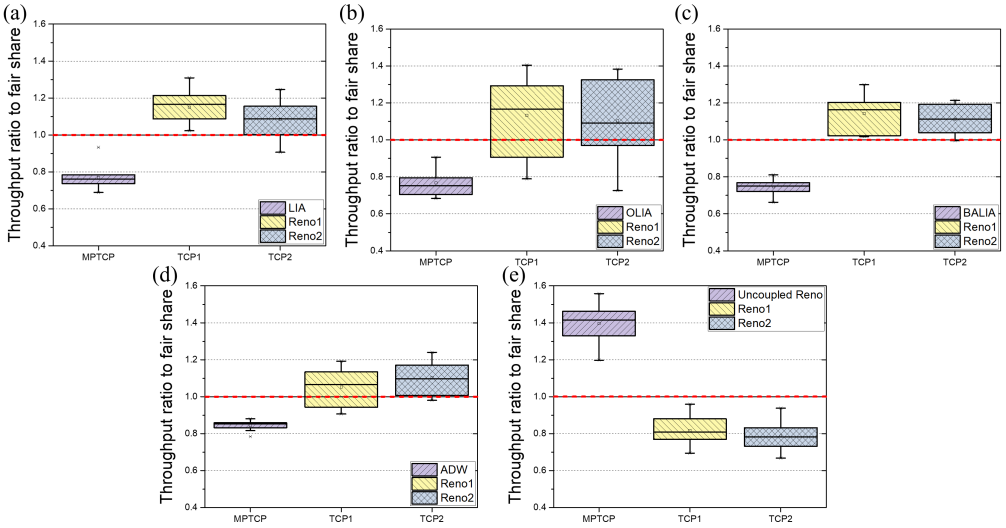


Figure 9. Evaluation results of Scenario 3: (**a**) linked increase adaptation (LIA); (**b**) optimized LIA (OLIA); (**c**) balanced LIA (BALIA); (**d**) adaptive decrease window (ADW); and, (**e**) uncoupled Reno.

Scenario 4 has the same topology as Scenario 3, but the bottleneck is changed to a nonshared bottleneck link. The bandwidth of each bottleneck link was changed to place a larger load on the nonshared bottleneck link that is located at the front. The LIA and OLIA algorithms exhibit similar results, but the BALIA algorithm demonstrates a reduced throughput ratio when compared to that of Scenario 2 (Figure 10a–c). Conversely, in Figure 10e, the uncoupled Reno takes up more bandwidth in Scenario 4. The proposed ADW-BALIA algorithm not only exhibits the best fairness, but also significantly improves the throughput when compared to the existing MPTCP algorithms.

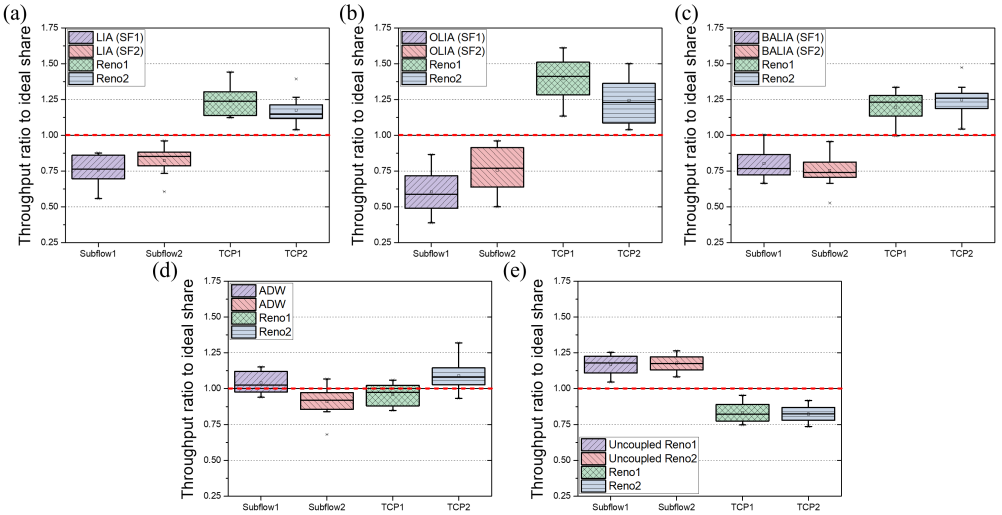


Figure 10. Evaluation results of Scenario 4: (**a**) linked increase adaptation (LIA); (**b**) optimized LIA (OLIA); (**c**) balanced LIA (BALIA); (**d**) adaptive decrease window (ADW); and, (**e**) uncoupled Reno.

Scenario 5 assumes a combination of Scenarios 3 and 4, where three bottleneck links exist simultaneously. New single-path TCP hosts are connected to a shared bottleneck that is located at the downstream, and the same load is applied to the three bottleneck links. Figure 11 presents the throughput ratio for Scenario 5. The existing MPTCP algorithms shown in Figure 11a–c have a low throughput ratio of 0.7 or less due to competition on consecutive bottleneck links. Although the ADW-BALIA algorithm does not reach the ideal throughput ratio, it accounts for about 80% of the ideal bandwidth, which results in

improved throughput (Figure 11d). For the uncoupled Reno, the single-path TCP flows are overpowered, resulting in very unfair results.

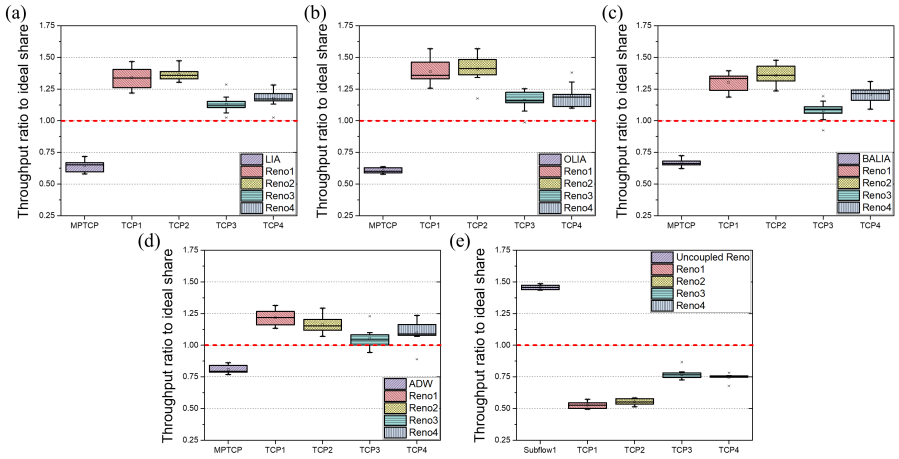


Figure 11. Evaluation results of Scenario 5: (**a**) linked increase adaptation (LIA); (**b**) optimized LIA (OLIA); (**c**) balanced LIA (BALIA); (**d**) adaptive decrease window (ADW); and, (**e**) uncoupled Reno.

4.2.2. Fairness Index

Figures 7–11 demonstrate how close each flow is to the ideal throughput, and Figure 12 presents a fairness index between the competing flows. The fairness between the MPTCP flows and TCP flows is calculated whlie using *Jain's fairness index* [46]:

$$Jain's\ Fairness\ Index = \frac{(\sum_{i=1}^{n} x_i)^2}{n \sum_{i=1}^{n} {x_i}^2}, \tag{5}$$

where n denotes the number of flows and x denotes the throughput of each flow.

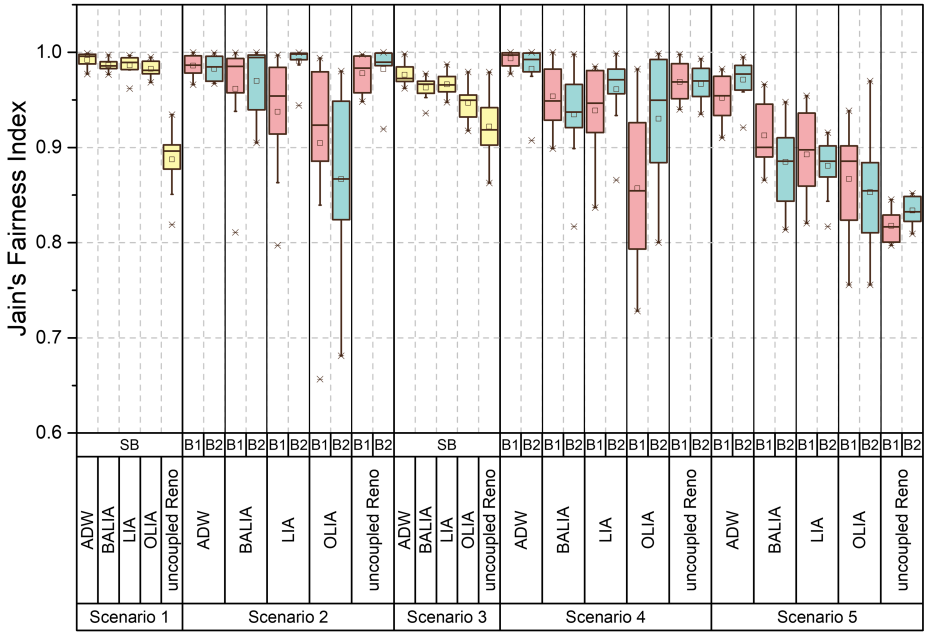


Figure 12. Comparison of Jain's fairness index for all scenarios.

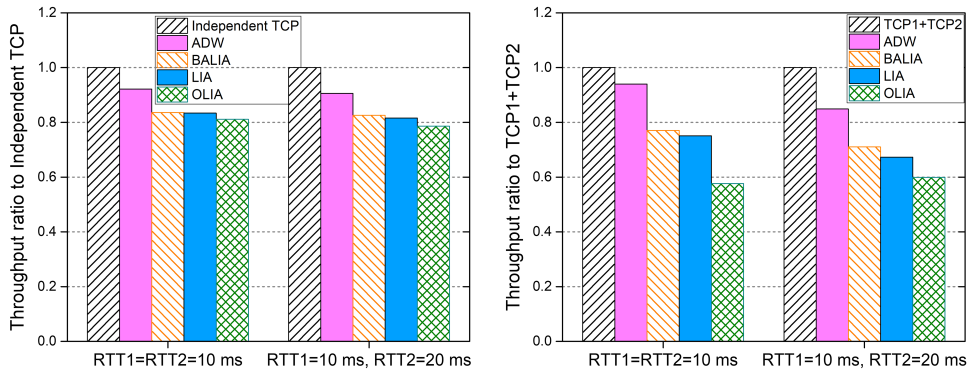
In the shared bottleneck scenario, the fairness for each throughput of two single-path TCP flows and the aggregate MPTCP throughput was compared, whereas the fairness between one MPTCP subflow and a single-path TCP flow in each nonshared bottleneck link was compared in the nonshared bottleneck scenario.

The uncoupled Reno algorithm exhibits a sufficient fairness index of 0.9 in Scenarios 2 and 4; however, in the shared bottleneck-related scenario, it overwhelms the single-path TCP flow, and the fairness index is reduced to approximately 0.9. In Scenarios 1 and 3, all of the congestion control algorithms, except the uncoupled Reno algorithm, exhibit an excellent fairness index and a small deviation. However, the fairness index of all MPTCP algorithms except the proposed ADW-BALIA algorithm decreases in Scenario 2 and 4, and the decrease in the OLIA algorithm is most noticeable. Moreover, their deviation also substantially increases. In the last scenario, the fairness index of all existing coupled congestion control algorithms is greatly reduced, which results in a throughput imbalance between the MPTCP flows and single-path TCP flows. Conversely, the ADW-BALIA algorithm uses the bandwidth quite fairly when each flow competes on the bottleneck link, showing the best fairness index with a small deviation.

4.2.3. Different Round-Trip Times

Because the ADW-BALIA algorithm proposed in this paper observes the RTT fluctuation of each subflow, the operation must be checked in situations in which the subflows have different delays. Figure 13 displays the throughput ratio when two MPTCP subflows have the same RTT and different RTTs in the topology of Scenarios 1 and 2.

In the shared bottleneck scenario result shown in Figure 13a, all of the MPTCP algorithms guarantee the fairness goal of MPTCP, regardless of whether the RTT of the path is different. In particular, the ADW algorithm has improved throughput when compared to the existing MPTCP algorithm without damaging the TCP flow. Figure 13b demonstrates the results of dividing the aggregate MPTCP throughput by two single TCP throughputs when the two paths have different delays in a nonshared bottleneck scenario. Unlike the shared bottleneck scenario, the throughput changes slightly, due to differences in delays between paths. The throughput of all MPTCP algorithms, except OLIA, is reduced due to the difference in the RTT of the two paths. The ADW algorithm that is designed based on the BALIA algorithm also exhibits a decrease in throughput, but has the highest throughput among the MPTCP algorithms.



(a) Shared bottleneck link (b) Nonshared bottleneck link

Figure 13. Throughput ratio for Scenarios 1 and 2 according to RTT difference.

4.2.4. Competition with the CUBIC Congestion Control Algorithm

All of the MPTCP congestion control algorithms implemented in Linux are designed based on TCP Reno, but, currently, the default TCP congestion control algorithm in Linux is CUBIC [22]. The CUBIC was implemented based on the proposed binary increment congestion (BIC) control algorithm to improve Reno's slow congestion window increase mechanism [47]. However, the BIC was difficult to analyze due to its complex design, and there was a problem of RTT fairness, in which flows with a short RTT occupied considerable bandwidth. Therefore, CUBIC was proposed to solve the above problems, and it has been designated as the default TCP congestion control algorithm in Linux since 2006.

Therefore, we performed an experiment in which the proposed ADW algorithm competes with the CUBIC flows. In the basic topologies of Figure 6a,c, the aggregate MPTCP throughput ratio was calculated when competing with the CUBIC flows, divided by its fair share. For comparison, Figure 14 presents the competition results with TCP Reno flows in each scenario. The CUBIC flows occupy more bottleneck bandwidth than the Reno flows in both scenarios. Therefore, all of the MPTCP algorithms show a lower throughput ratio when competing with CUBIC flows than when competing with Reno flows. In a nonshared bottleneck scenario, the ADW algorithm has a 20% higher average throughput ratio than the BALIA and LIA algorithms and it takes up sufficient bandwidth, even if it competes with the CUBIC flows.

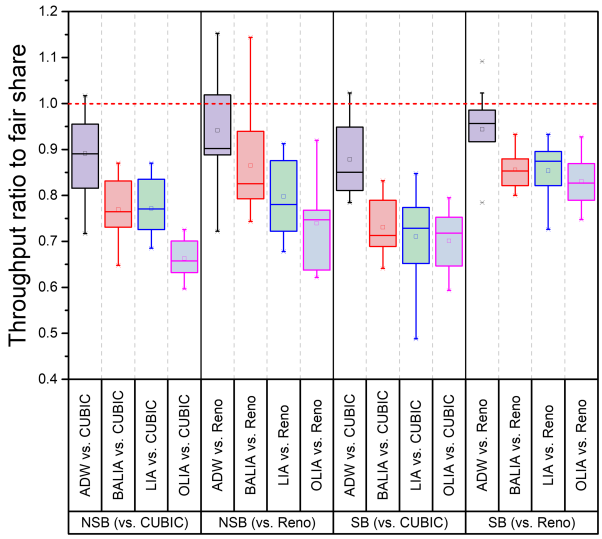


Figure 14. Throughput ratio to fair share when MPTCP competes with the CUBIC.

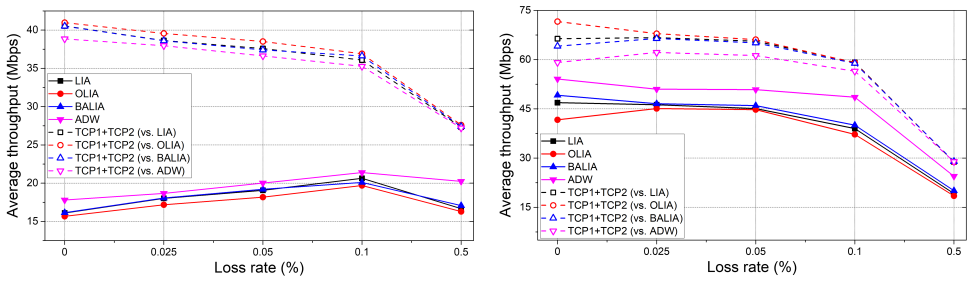
In a shared bottleneck scenario, the existing MPTCP congestion control algorithm has a throughput ratio of higher than 0.8 when competing with the Reno flow, but it exhibits a throughput ratio of lower than 0.8 in competition with the CUBIC flows. This outcome indicates that the MPTCP connection cannot guarantee enough throughput when competing with a single-path CUBIC flow on a shared bottleneck link. In contrast, the ADW algorithm occupies sufficient bandwidth with a throughput ratio of 0.8 or more in a shared bottleneck scenario.

4.2.5. Lossy Environment

All of the introduced experiments were conducted in a lossless environment. The MPTCP congestion control algorithm, which inherits the loss-based TCP congestion control that performs congestion control using packet loss as a congestion signal, may experience performance degradation in a lossy link. Moreover, the RTT fluctuation measurement may not be accurate in a link with frequent loss because the proposed ADW-BALIA algorithm considers the RTT fluctuation of each subflow. Therefore, we evaluated the proposed ADW algorithm performance in an environment, where additional loss exists besides the packet loss due to congestion events.

Figure 15 illustrates the change in the average throughput with increasing random loss rates for shared and nonshared bottleneck links. The throughput sum of two single-path Reno flows and the aggregate MPTCP throughput are plotted together. In a shared bottleneck scenario presented in Figure 15a, as the loss rate increases ($\leq$0.1%), the average throughput of the MPTCP connection gradually increases, and the total throughput of the single-path TCP gradually decreases. At a loss rate of 0.1%, the MPTCP congestion control slightly exceeds its fair share ($\simeq$20 Mbps in lossless), but it does not significantly impair fairness. At a high loss rate of 0.5%, the throughput reduction of the single-path

TCP is noticeable, significantly reducing the overall throughput (MPTCP and two TCP flows). The proposed ADW-BALIA algorithm has a higher throughput than the existing MPTCP algorithm for all loss rates, and the gap is wider, especially at a loss rate of 0.5%. Moreover, at a 0.5% loss rate, the ADW-BALIA algorithm occupies excess bandwidth without harming the single-path TCP, leading to increased throughput. The nonshared bottleneck experiment presented in Figure 15b also reveals results that are similar to the shared bottleneck experiment. In both scenarios, the ADW-BALIA algorithm maintains adequate fairness with the single-path TCP and improves the throughput.



(a) Shared bottleneck link (b) Nonshared bottleneck link

Figure 15. Average throughput by loss rate in Scenarios 1 and 2.

5. Conclusions

In this paper, we proposed the ADW for BALIA congestion control algorithm in order to improve the throughput in a nonshared bottleneck while maintaining TCP friendliness in a shared bottleneck. The existing MPTCP congestion control algorithms fairly compete with the single-path TCP in a shared bottleneck, but have not achieved the maximum throughput in a nonshared bottleneck. The congestion window decrease mechanism of the ADW algorithm is adjusted based on the similarity between the subflows estimated through the delay information to achieve the maximum throughput. If the delay fluctuations between subflows are similar, then the algorithm operates like the existing BALIA congestion control algorithms and it attenuates the reduction in the cwnd according to the difference in delay fluctuations. The proposed ADW-BALIA algorithm was evaluated by conducting Mininet-based emulation experiments. The experimental results reveal that the proposed ADW-BALIA algorithm coexists well with the single-path TCP in a shared bottleneck and that the throughput increases by 20% as compared to the default congestion control of the Linux MPTCP. In addition, we revealed that the proposed algorithm effectively works in both simple and complex topologies with two or more bottlenecks. In all scenarios, the ADW-BALIA algorithm had the highest and most stable fairness index. The experiments on different RTT paths, the competition with CUBIC flows, and the lossy links confirm that the ADW-BALIA algorithm improved the throughput when compared to the existing MPTCP congestion control algorithms. In future work, we will investigate the performance of ADW-BALIA on buffers while using different active queue management algorithms.

Author Contributions: G.-H.K. proposed the idea, conducted the experiments, and wrote the manuscript. Y.-J.S. contributed by setting up the additional experiment environment. I.M. contributed by providing the feedback for the evaluation results. Y.-Z.C. supervised the entire research. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Ministry of Education, 2018R1A6A1A03025109, and was funded by the Korea government (MSIT), 2019R1A2C1006249.

Acknowledgments: This research was supported in part by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. NRF-2018R1A6A1A03025109), and by the NRF grant funded by the Korea government (MSIT) (No. NRF-2019R1A2C1006249).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

TCP	Transmission control protocol
MPTCP	Multipath TCP
ADW	Adaptive decrease window
cwnd	Congestion window
AIMD	Additive increase/multiplicative decrease
LIA	Linked increase adaptation
OLIA	Optimized LIA
BALIA	Balanced LIA
BBR	Bottleneck bandwidth and round-trip time
RTT	Round-trip time
SF	Subflow
SB	Shared bottleneck
NSB	Nonshared bottleneck
BIC	Binary increment congestion
AQM	Active queue management

References

1. Postel, J. Transmission Control Protocol. 1981. Available online: https://tools.ietf.org/html/rfc793 (accessed on 25 January 2021).
2. Floyd, S.; Henderson, T. The New Reno Modification to TCP's Fast Recovery Algorithm. 1999. Available online: https://tools.ietf.org/html/rfc2582 (accessed on 25 January 2021).
3. Allman, M.; Paxson, V.; Blanton, E. TCP Congestion Control. 2009. Available online: https://tools.ietf.org/html/rfc5681 (accessed on 25 January 2021).
4. Jacobson, V. Congestion avoidance and control. *ACM SIGCOMM Comput. Commun. Rev.* **1988**, *18*, 314–329. [CrossRef]
5. Long Term Evolution (LTE): A Technical Overview. 2010. Available online: https://www.3g4g.co.uk/Lte/LTE_WP_0706_Motorola.pdf (accessed on 25 January 2021).
6. Agiwal, M.; Roy, A.; Saxena, N. Next Generation 5G Wireless Networks: A Comprehensive Survey. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 1617–1655. [CrossRef]
7. Chen, Y.-C.; Lim, Y.; Gibbens, R.; Nahum, E.; Khalili, R.; Towsley, D. A Measurement-based Study of Multipath TCP Performance over Wireless Networks. In Proceedings of the Internet Measurement Conference (IMC), Barcelona, Spain, 23–25 October 2013; pp. 455–468.
8. Xu, C.; Zhao, J.; Muntean, G.-M. Congestion control design for multipath transport protocols: A survey. *IEEE Commun. Surv. Tutor.* **2016**, *18*, 2948–2969. [CrossRef]
9. Ford, A.; Raiciu, C.; Handley, M.; Barre, S.; Iyengar, J. Architectural Guidelines for Multipath TCP Development. 2011. Available online: https://tools.ietf.org/html/rfc6182 (accessed on 25 January 2021).
10. Ford, A.; Raiciu, C.; Handley, M.; Bonaventure, O. TCP Extensions for Multipath Operation with Multiple Addresses. 2013. Available online: https://tools.ietf.org/html/rfc6824 (accessed on 25 January 2021).
11. Khalili, R.; Gast, N.; Popovic, M.; Le Boudec, J. MPTCP is not Pareto-Optimal: Performance Issues and a Possible Solution. *IEEE/ACM Trans. Netw.* **2013**, *21*, 1651–1654. [CrossRef]
12. Scharf, M.; Ford, A. Multipath TCP (MPTCP) Application Interface Considerations. 2013. Available online: https://tools.ietf.org/html/rfc6897 (accessed on 25 January 2021).
13. Raiciu, C.; Wischik, D.; Handley, M. *Practical Congestion Control for Multipath Transport Protocols*; Technical Report; University College London: London, UK, 2009.
14. Wischik, D.; Handley, M.; Braun, M.B. The Resource Pooling Principle. *ACM SIGCOMM Comput. Commun. Rev.* **2008**, *38*, 47–52. [CrossRef]
15. Handley, M.; Padhye, J.; Floyd, S. TCP Congestion Window Validation. 2000. Available online: https://tools.ietf.org/html/rfc2861 (accessed on 25 January 2021).
16. Chiu, D.; Jain, R. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Comput. Netw. ISDN Syst.* **1989**, *17*, 1–14. [CrossRef]
17. Shakkottai, S.; Hollot, C.V.; Srikant, R.; Towsley, D. *Overlay TCP for Multi-Path Routing and Congestion Control*; Ens-Inria Arc-TCP Workshop: Paris, France, 5–7 November 2004.
18. Raiciu, C.; Handley, M.; Wischik, D. Coupled Congestion Control for Multipath Transport Protocols. 2011. Available online: https://tools.ietf.org/html/rfc6356 (accessed on 25 January 2021).
19. Peng, Q.; Walid, A.; Hwang, J.; Low, S.H. Multipath TCP: Analysis, design, and implementation. *IEEE/ACM Trans. Netw.* **2016**, *24*, 596–609. [CrossRef]
20. Floyd, S.; Mahdavi, J. TCP-Friendly Unicast Rate-Based Flow Control. Technical Note Sent to end2end-Interest Mailing List. 1997. Available online: http://www.psc.edu/networking/papers/tcpfriendly.html (accessed on 25 January 2021).

21. Ferlin, S.; Alay, O.; Dreibholz, T.; Hayes, D.A.; Welzl, M. Revisiting congestion control for multipath TCP with shared bottleneck detection. In Proceedings of the IEEE International Conference on Computer Communications, Chengdu, China, 14–17 October 2016; pp. 1–9.
22. Ha, S.; Rhee, I.; Xu, L. CUBIC: A new TCP-friendly high speed TCP variant. *ACM SIGOPS Oper. Syst. Rev.* **2008**, *42*, 5. [CrossRef]
23. Cardwell, N.; Cheng, Y.; Gunn, C.S.; Yeganeh, S.H.; Jacobson, V. BBR: Congestion-based congestion control. *ACM Queue* **2016**, *14*, 20–53. [CrossRef]
24. Honda, M.; Nishida, Y.; Eggert, L.; Sarolahti, P.; Tokuda, H. Multipath Congestion Control for Shared Bottleneck. 2009. Available online: https://eggert.org/papers/2009-pfldnet-mpath-cc.pdf (accessed on 25 January 2021).
25. Cao, Y.; Xu, M.; Fu, X. Delay-based congestion control for multipath TCP. In Proceedings of the IEEE International Conference Network Protocols (ICNP), Austin, TX, USA, 30 October–2 November 2012; pp. 1–10.
26. Kimura, B.Y.L.; Loureiro, A.A.F. MPTCP Linux Kernel Congestion Controls. *arXiv* **2018**, arXiv:1812.03210.
27. Nguyen, K.; Kibria, M.G.; Ishizu, K.; Kojima, F.; Sekiya, H. An Evaluation of Multipath TCP in Lossy Environment. In Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops), Kyoto, Japan, 11–15 March 2019; pp. 573–577.
28. Zhang, S.; Lei, W.; Zhang, W.; Guan, Y.; Li, H. Congestion Control and Packet Scheduling for Multipath Real Time Video Streaming. *IEEE Access* **2019**, *7*, 59758–59770. [CrossRef]
29. Zhu, T.; Qin, X.; Chen, L.; Chen, X.; Wei, G. wBBR: A Bottleneck Estimation-Based Congestion Control for Multipath TCP. In Proceedings of the IEEE Vehicular Technology Conference (VTC-Fall), Chicago, IL, USA, 27–30 August 2018; pp. 1–5.
30. Han, J.; Xue, K.; Xing, Y.; Hong, P.; Wei, D.S. Measurement and Redesign of BBR-based MPTCP. In Proceedings of the ACM SIGCOMM Conference Posters and Demos, Beijing, China, 19–23 August 2019; pp. 75–77.
31. Mahmud, I.; Lubna, T.; Song, Y.-J.; Cho, Y.-Z. Coupled Multipath BBR (C-MPBBR): A Efficient Congestion Control Algorithm for Multipath TCP. *IEEE Access* **2020**, *8*, 165497–165511. [CrossRef]
32. Mahmud, I.; Kim, G.-H.; Lubna, T.; Cho, Y.-Z. BBR-ACD: BBR with Advanced Congestion Detection. *Electronics* **2020**, *9*, 136. [CrossRef]
33. Kim, G.-H.; Cho, Y.-Z. Delay-Aware BBR Congestion Control Algorithm for RTT Fairness Improvement. *IEEE Access* **2020**, *8*, 4099–4109. [CrossRef]
34. Song, Y.-J.; Kim, G.-H.; Cho, Y.-Z. BBR-CWS: Improving the Inter-Protocol Fairness of BBR. *Electronics* **2020**, *9*, 862. [CrossRef]
35. Hassayoun, S.; Iyengar, J.; Ros, D. Dynamic Window Coupling for Multipath Congestion Control. In Proceedings of the IEEE International Conference on Network Protocols (ICNP), Vancouver, BC, Canada, 17–20 October 2011; pp. 341–352.
36. Callegari, C.; Giordano, S.; Pagano, M.; Pepe, T. A Survey of Congestion Control Mechanisms in Linux TCP. In Proceedings of the International Conference on Distributed Computer and Communication Networks (DCCN), Moscow, Russia, 7–10 October 2013; pp. 28–42.
37. Wei, W.; Wang, Y.; Xue, K.; Wei, D.S.L.; Han, J.; Hong, P. Shared bottleneck detection based on congestion interval variance measurement. *IEEE Commun. Lett.* **2018**, *22*, 2467–2470. [CrossRef]
38. Zhang, M.; Lai, J.; Krishnamurthy, A.; Peterson, L.L.; Wang, R.Y. A transport layer approach for improving end-to-end performance and robustness using redundant paths. In Proceedings of the USENIX Annual Technical Conference (ATC), Carlsbad, CA, USA, 11–13 July 2004; pp. 99–112.
39. Kim, M.S.; Kim, T.; Shin, Y.; Lam, S.S.; Powers, E.J. A wavelet-based approach to detect shared congestion. *IEEE/ACM Trans. Netw.* **2008**, *16*, 763–776.
40. Yousaf, M.M.; Welzl, M. On the accurate identification of network paths having a common bottleneck. In Proceedings of the ACM SIGCOMM Conference, Seattle, WA, USA, 17–22 August 2008.
41. Wei, W.; Xue, K.; Han, J.; Wei, D.S.L.; Hong, P. Shared Bottleneck-Based Congestion Control and Packet Scheduling for Multipath TCP. *IEEE/ACM Trans. Netw.* **2020**, *28*, 653–666. [CrossRef]
42. Floyd, S.; Ramakrishnan, D.K.K.; Black, D.L. The Addition of Explicit Congestion Notification (ECN) to IP. 2001. Available online: https://tools.ietf.org/html/rfc3168 (accessed on 25 January 2021).
43. Lantz, B.; Heller, B.; McKeown, N. A network in a laptop: Rapid prototyping for software-defined networks. In Proceedings of the ACM SIGCOMM Workshop Hot Topics in Networks, Monterey, CA, USA, 20–21 October 2010.
44. Hemminger, S.; Ludovici, F.; Paul, H. NetEm—Network Emulator. Available online: http://manpages.ubuntu.com/manpages/bionic/man8/tc-netem.8.html/ (accessed on 24 September 2020).
45. Dugan, J. Iperf3—Perform Network Throughput Tests. Available online: http://manpages.ubuntu.com/manpages/bionic/en/man1/iperf3.1.html/ (accessed on 24 September 2020).
46. Jain, R.; Chiu, D.M.; Hawe, W.R. *A Quantitative Measure of Fairness and Discrimination for Resource Allocation in Shared Computer System*; Tech. Rep. TR-301; Digital Equipment Corporation: Maynard, MA, USA, 1984.
47. Xu, L.; Harfoush, K.; Rhee, I. Binary increase congestion control (BIC) for fast long-distance networks. In Proceedings of the IEEE INFOCOM, Hong Kong, China, 7–11 March 2004.