SDN and BBR Based Friendly Congestion Control for MPTCP

1st Ganlin Tang
Department of Computer Science
Kookmin University
Seoul, Korea
gltang@kookmin.ac.kr

2nd Sang-Chul Kim
Department of Computer Science
Kookmin University
Seoul, Korea
sckim7@kookmin.ac.kr

***Abstract*—MultiPath TCP (MPTCP) is a transport protocol that uses multiple network interfaces and IP addresses, allowing TCP data to be sent over different network paths at the same time. The Internet Engineering Task Force (IETF) has developed several congestion control algorithms and packet schedulers for MPTCP to improve fairness and increase the throughput of each subflow. However, congestion control algorithms that rely on the congestion window (CWND) have limitations because they cannot easily detect which link is the bottleneck or accurately measure the available bandwidth, making it difficult to maximize total throughput across all subflows. In recent years, pacing-based congestion control algorithms have been introduced to send packets at a controlled rate and better identify subflows that share the same bottleneck link, helping to improve overall throughput. BBR a pacing-based congestion control algorithm controls the sending rate based on bottleneck bandwidth estimated and round-trip propagation time. However, these pacing-based algorithms do not fully meet the friendliness requirements of MPTCP and cannot precisely determine whether subflows are on the same bottleneck link. To address these issues, this paper proposes a new MPTCP congestion control algorithm and packet scheduler that combine Software-Defined Networking (SDN) with the BBR algorithm which call FMBBR. The results of the experiment show that FMBBR not only maintains high throughput but also improves fairness of the network environment, and SDN enables MPTCP to identify the same bottleneck links accurately.**

***Index Terms*—Multipath TCP, BBR, Fairness, Congestion Control, Packet Scheduling**

I. Introduction

In recent years, the demand for high-performance and reliable network transmission has been increasing, but using a single link often faces many limitations. MultiPath TCP (MPTCP) [1], an extension of TCP, allows senders to use multiple IP addresses and ports to achieve more efficient data transmission. Each subflow works as an independent TCP connection, which improves transmission efficiency and enhances network stability, providing users with better quality of service.

TCP relies on congestion control algorithms to prevent network congestion. However, MPTCP contains multiple subflows that may share the same path, which makes it difficult to directly apply traditional TCP congestion control algorithms to MPTCP [2]. In recent years, several congestion control algorithms designed for MPTCP have been proposed, including LIA [3], BALIA [4], and OLIA [5]. These algorithms improve MPTCP throughput while maintaining fairness among subflows. However, to ensure fairness, they couple the throughput of all subflows, which limits the total throughput of MPTCP and prevents it from exceeding the throughput of the best single-path TCP connection [6].

Bottleneck Bandwidth and Round-trip Propagation Time (BBR) [7] is a congestion control algorithm that operates in multiple stages. It uses the round-trip time and the number of packets in flight to build a clear network model. By estimating the available bandwidth, the bandwidth-delay product, and the maximum number of packets that can be sent on the current path, BBR achieves high throughput while keeping queue pressure low. This makes BBR a promising approach to improve the transmission efficiency of MPTCP. In recent years, several coupled congestion control algorithms that combine BBR with MPTCP have been proposed [8]–[10].

MPTCP requires coupled control of multiple subflows, which makes it challenging to determine whether subflows are on the same path in complex network environments [11]. Software Defined Networking (SDN) [12] is an emerging technology that can help address this issue [13]. SDN uses the OpenFlow protocol [14] to separate the control plane from the data plane in routers. This separation provides precise routing information for each subflow, making it possible to accurately identify shared bottleneck links.

In this paper, we propose a friendly MPTCP coupled congestion control algorithm called Friendly Multipath BBR (FMBBR). FMBBR is based on SDN for shared bottleneck detection and BBR for congestion control. The algorithm keeps the high throughput benefits of BBR while accurately identifying groups of collinear subflows. It also reduces the negative impact on the network performance of other users. The main contributions of this paper are summarized as follows:

1) We propose an improved MPTCP congestion control algorithm called FMBBR, which combines SDN and BBR. FMBBR uses SDN to accurately identify all MPTCP subflows that share a bottleneck link. Once identified, these subflows gradually perform bandwidth probing in a coordinated manner. This approach protects the network quality for other users while improving the overall throughput of MPTCP.

2) We conducted simulation experiments using Mininet. The results show that, compared to directly using BBR for bandwidth probing, FMBBR achieves a higher initial probing bandwidth and reduces the negative impact on other users. In addition, we compared FMBBR with other MPTCP congestion control algorithms, and the results demonstrate that FMBBR provides better overall performance.

The remainder of this article is organized as follows. Section II introduces the background and related work. Section III describes the design of FMBBR. Section IV presents the performance evaluation. Finally, Section V concludes the study.

II. BACKGROUND

The Internet Engineering Task Force (IETF) defined three main design objectives for MPTCP [2]:

1) Improve Throughput: The throughput of MPTCP should be at least as high as the throughput of the best single path among all available paths.
2) Do No Harm: The capacity used by MPTCP on any path should not be greater than the capacity used by a regular TCP connection.
3) Balance Congestion: When the first two conditions are satisfied, MPTCP should help relieve network congestion and maintain balanced transmission efficiency.

The first goal is the main reason for designing MPTCP. The second goal aims to ensure fairness in the Internet environment and improve overall service quality. The third goal is based on the resource pooling principle [15]. As a result, most MPTCP-related algorithms focus on congestion control and shared bottleneck detection.

A. Bottleneck Detection

[11] points out that current bottleneck detection methods often rely on packet loss and latency to identify shared bottlenecks. However, these signals are rare and unreliable in well-performing network environments. To address this issue, they propose using One-Way Delay (OWD) as the main metric, combined with skewness, variability, and key frequency, to detect shared bottlenecks as described in [16]. However, their approach requires adding new MPTCP timestamps to the kernel, which is less flexible than using modules or eBPF. In addition, the algorithm depends heavily on time synchronization. In real-world Internet environments, time synchronization errors can greatly affect the accuracy of the measurements.

[17] proposes a new method for shared bottleneck detection based on Explicit Congestion Notification (ECN). This method uses the birth-death Markov model [18] to calculate an optimal sampling period D. Detection is then performed by analyzing the arrival times of ECN packets within this period. However, although ECN adoption has grown, many devices still do not fully support it. In addition, in complex network topologies with multiple bottlenecks and queues, ECN-based collinearity detection can easily lead to misjudgments or missed detections.

B. Congestion Control

To meet the design requirements, existing congestion control algorithms such as LIA [3], BALIA [4], and OLIA [5] aim to achieve relative fairness. However, they prevent MPTCP from fully using the available bandwidth of subflows, regardless of whether those subflows share the same bottleneck path. In addition, congestion algorithms based on congestion windows and packet loss detection mainly rely on Additive Increase Multiplicative Decrease (AIMD) to control the congestion window (CWND). When packet loss occurs, a subflow's CWND is typically reduced by half. As a result, packet loss in one subflow can also affect other subflows under these algorithms. Consequently, these approaches focus more on satisfying the second design rule but often fail to meet the first rule.

Google's BBR [7] algorithm is a rate-based congestion control method that uses a built-in network model to improve bandwidth utilization and reduce latency. Since BBR controls both the congestion window (CWND) and the transmission rate, it can maintain high transmission efficiency even when packet loss occurs.

[19] implements a simple multipath BBR by using BBR as the congestion control algorithm for each subflow. They evaluate the transmission performance of MPTCP under different packet loss scenarios. However, their approach does not consider the principle of fairness between subflows.

[10] proposes a coupling-based method to control the congestion of all subflows. This approach allocates the bandwidth of the best-performing subflow to all subflows according to the detected bandwidth of each one. Although this method effectively maintains fairness in MPTCP, it still limits the actual available bandwidth of subflows in real network conditions.

[9] uses RTT to identify subflows that share the same bottleneck. Based on this detection, the algorithm only limits the bandwidth of subflows on the same bottleneck path. As a result, the total bandwidth of these subflows does not exceed the maximum bandwidth of the best-performing subflow on the current path, while still ensuring efficient bandwidth utilization.

BBR operates in several stages: STARTUP, DRAIN, PROBEBW, and PROBERTT. During these stages, BBR uses two key parameters, pacing-gain and cwnd-gain, to control the sending rate and the size of the congestion window (CWND).

For ordinary TCP, BBR executes these stages in order. In the STARTUP phase, BBR sets the pacing-gain to 2/ln2 to quickly fill the link and estimate the maximum available bandwidth. This approach works well for single-path TCP. However, in MPTCP, when multiple subflows pass through the same bottleneck link, starting the STARTUP phase of BBR at the same time can lead to severe network congestion [20].

Therefore, although [9] follows the second principle of the MPTCP design guidelines, the special bandwidth detection method used by BBR often fails to detect enough bandwidth during the initial detection phase. As a result, subflows that share the same bottleneck link may not use the available bandwidth efficiently.

At the same time, during the STARTUP stage, subflows that share the same bottleneck link can cause a significant negative impact on other network users during the initial detection process, even though their bandwidth will later be limited in the PROBE_BW stage.

III. System Design Detials

In this section, we introduce FMBBR, which is designed to detect subflows that share the same bottleneck and achieve high initial bandwidth detection under friendly conditions.

A. Overview

To achieve high throughput with the BBR algorithm while following the design principles of MPTCP, FMBBR introduces a user-friendly initialization phase that leverages an SDN-based feedback mechanism to identify bottleneck subflows. After detection, it applies a congestion control algorithm to allocate optimal bandwidth to each subflow during transmission. This approach reduces the negative impact on other users and improves MPTCP's overall utilization of network resources.

B. SDN Based Bottleneck Detection

We use the controller to capture SYN and SYN|ACK packets sent by MPTCP during the initial phase to collect data. A small server runs inside the controller. Once the MPTCP sender confirms that all subflows are established, it retrieves the data from the server.

MPTCP uses TCP options [1] to control its behavior, as shown in TABLE I. To accurately capture MPTCP SYN packets, the controller only collects SYN and SYN|ACK packets that contain the MP-Capable and MP-JOIN options.

To avoid misjudgments caused by other MPTCP connections in the network, we extract the receiver's key from SYN|ACK packets containing MP-Capable and use it to calculate a unique token. For each subflow that joins the connection, its MPTCP option must include both MP-JOIN and the same token. This process allows us to accurately gather path information for all subflows. By checking whether these subflows pass through the same router, we can identify which subflows share the same bottleneck link.

TABLE I: MPTCP Option Subtypes

Value	Symbol	Name
0x0	MP_CAPABLE	Multipath Capable
0x1	MP_JOIN	Join Connection
0x2	DSS	Data Sequence Signal
0x3	ADD_ADDR	Add Address
0x4	REMOVE_ADDR	Remove Address
0x5	MP_PRIO	Change Subflow Priority
0x6	MP_FAIL	Fallback
0x7	MP_FASTCLOSE	Fast Close

C. BBR Based Congestion Control

Before all subflows are established, each subflow maintains its default minimum congestion window (cwnd) for data transmission. During this stage, we record the minimum RTT of each subflow.

After retrieving the collinear subflow data from the SDN controller, the subflows are instructed to enter the STARTUP phase in order, based on their minimum RTT and collinearity status.

To reduce the total time spent in the STARTUP phase, subflows on different paths are initiated at the same time. For subflows that share the same bottleneck path, they are first sorted by their minimum RTT and then started sequentially.

To prevent incomplete detection of subflows on the same bottleneck path and to minimize the impact on other network users, only one subflow on the same bottleneck path is allowed to be in the STARTUP phase at any given time.

After all subflows complete the STARTUP phase, we record the maximum bandwidth obtained by each subflow i during the detection process as $prob\_bwi$. We then determine the MAX_BW for the entire set of subflows sharing the same bottleneck link S. The calculation of MAX_BW can be expressed as:

$$MAX\_BW \leftarrow \max_{i \in S} prob\_bw_i \tag{1}$$

After obtaining MAX_BW, we allocate a bandwidth limit $limitbw_i$ to each subflow sub_i based on the order of their RTT values. The allocation for each subflow sub_i can be described as:

$$limitbw_i = \begin{cases} prob\_bw_i, & MAX\_BW \geq prob\_bw_i \\ MAX\_BW, & MAX\_BW < prob\_bw_i \end{cases} \tag{2}$$

$$MAX\_BW = MAX\_BW - limitbw_i \tag{3}$$

Algorithm 1 Give the pseudocode for the description.

D. Initial Redundant Packet Scheduler

For BBR-based MPTCP, when a subflow enters the STARTUP phase, both the $pacing\_gain$ and $cwnd\_gain$ are set to $2/ln2$. During this phase, the subflow needs to send a large number of packets to probe the available link bandwidth. Therefore, we prioritize allocating packets to the subflow in this state. However, to avoid excessive packet loss during the probing process and to prevent overuse of $RWND$, which could interfere with the normal transmission of other non-collinear subflows, we design a simple redundant packet scheduling mechanism called Initial Redundant Packet Scheduler (IRPS). This scheduler specifically manages subflows in the STARTUP phase.

After all subflows complete the STARTUP stage, we schedule packets based on the smoothed round-trip time (SRTT) of each subflow. Subflows with lower SRTT are given higher priority to improve transmission efficiency and reduce overall latency. Algorithm 2 shows the pseudocode for this scheduling process.

Algorithm 1 FMBBR Congestion Control Algorithm

Step 1: Build up all subflows and collect min rtt
if All subflows build up done **then**
  Go to Step 2
  for Each subflow i get a new ack **do**
    if $min\_rtt_i < current\_rtt_i$ **then**
      $min\_rtt_i = current\_rtti$
    end if
  end for
end if
Step 2: Startup each subflow
Get bottleneck link subflows from controller to S
$S \leftarrow \text{sort}(S, \text{key} = min\_rtt, \text{order} = ascending)$
for each $sub_i \notin S$ **do**
  $sub_i.bbr\_mode = STARTUP$
end for
$index = 0$ ▷ subflows in set S get in STARTUP by $index$
for each $sub_j \in S$ **do**
  if $sub_j = S[index]$ **then**
    $sub_j.bbr\_mode = STARTUP$
  else if $sub_j.bbr\_mode = PROBE\_BW$ and $sub_j = S[index]$ **then**
    $index = index + 1$
  else if $sub_j.bbr\_mode = PROBE\_BW$ and $sub_j \neq S[index]$ **then**
    $sub_j.cwnd = 4$ ▷ According to BBR parameter
  end if
end for
Step 3: Bandwidth limited
$MAX\_BW \leftarrow \max_{i \in S} prob\_bw_i$
for $sub_i \in S$ **do**
  if $MAX\_BW > prob\_bw_i$ **then**
    $limitbw_i = prob\_bw_i$
    $MAX\_BW = MAX\_BW - limitbw_i$
  else
    $limitbw_i = MAX\_BW$
    $MAX\_BW = 0$
  end if
end for

Algorithm 2 Initial Redundant Packet Scheduler

Step 1: Startup Not Finished
for Each sub_i **do**
  if sub_i in STARTUP stage **then**
    Schedule redundant packet to sub_i
  else
    Schedule normal packet to sub_i
  end if
end for
Step 2: Startup Finished
$bestsk \leftarrow \arg\min_{i \in N} subi.srtt$
Schedule normal packet to $bestsk$

IV. PERFORMANCE ANALYSIS

A. Simulation Setting

We implement FMBBR and IRPS as Linux kernel modules in MPTCP v0.96, running on Linux kernel version 5.4.230. To evaluate their performance, we build a simple network topology using the Mininet simulator [21] and use RYU [22] as the SDN controller. The BBR version is bbr_v1 and the algorithm follows the default parameter setting. We implement random packet loss model in the topology. Fig. 1 shows the topology of the evaluated bottleneck link, while TABLE II lists the relevant parameters of the link, S is the set of links that share the same bottleneck.

TABLE II: Topo Parameters

Item	Shared Bottleneck	Non-shared bottleneck
Bandwidth [Mbps]	$0.8 \times \sum_{i \in S} \text{bw}_i$	10-25
Delay [ms]	15	10-25
Loss [%]	0.1	0.05
Router buffer [BDP]	1	1

To simplify the control of each subflow's path, we use fullmesh as the path manager. The receiver is directly connected to an access switch, referred to as R5, where both delay and queue size are negligible. Therefore, no bottleneck issues occur at this point in the network.

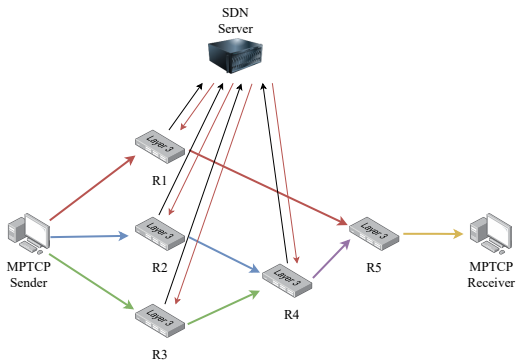


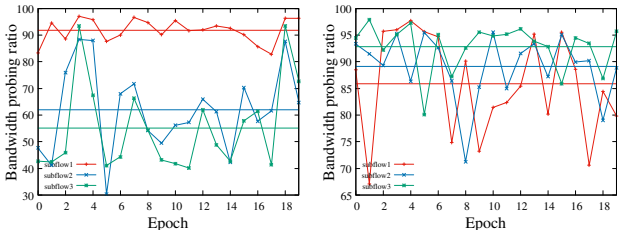
Fig. 1: Mininet Topo

B. Performance Evaluation

1) Initial Probed Maximum Bandwidth: We first compare the initial bandwidth measurements obtained using the FMBBR congestion control algorithm and the standard BBR algorithm.

Algorithm 1 shows that the initial maximum bandwidth detected for the same bottleneck directly affects the subsequent bandwidth allocation limit. Therefore, a higher initially measured available bandwidth for the same bottleneck leads to better transmission performance in later stages.

In Fig. 2, subflow 1 operates on an independent link, while subflow 2 and subflow 3 share the same bottleneck link. The results show that when BBR is used as the congestion control algorithm, only subflow 1 achieves a high bandwidth detection value. In contrast, subflow 2 and subflow 3, which pass through



(a): BBR (b): FMBBR

Fig. 2: Bandwidth Probing Ratio

the shared bottleneck, detect only a small amount of available bandwidth.

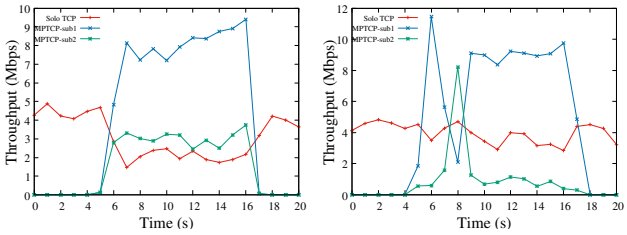
In contrast, when using FMBBR, all subflows are able to detect high bandwidth values. This is because subflows that share the same bottleneck link are detected sequentially, ensuring that they do not interfere with each other during the detection process.

2) Other User-friendliness: To verify the friendliness of FMBBR toward other users sharing the bottleneck link, we established an independent TCP connection between R4 and R5 with a fixed bandwidth of 5 Mbps. This connection also uses BBR as its congestion control algorithm.

We evaluated the overall user friendliness of FMBBR and compared it with standard BBR. In addition, we examined the impact of FMBBR's gradual STARTUP phase by comparing its performance with and without this mechanism.

(a): As shown in Fig. 3(a), when MPTCP uses BBR as its congestion control algorithm, it quickly consumes bandwidth. This causes the independent TCP connection to decrease its bandwidth, which would be influence to other users in real-world network scenarios.

As shown in Fig. 3(b), which illustrates the changes in bandwidth over time, individual TCP connections are affected by MPTCP's traffic preemption during transmission. However, they are still able to keep a high bandwidth.

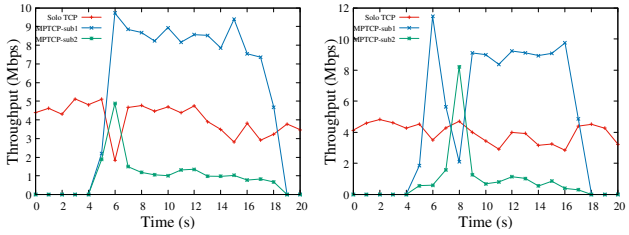


(a): BBR (b): FMBBR

Fig. 3: Comparison of BBR and FMBBR Friendliness

(b): Fig. 4(a) shows the scenario where FMBBR is used as the congestion control algorithm, but the collinear subflows are not gradually entered into the STARTUP phase. As seen in the figure, when the collinear subflows simultaneously enter the STARTUP phase, the independent TCP connection experiences a sharply bandwidth decrease, which occur around the 6-second mark.

In contrast, Fig. 4(b) shows the scenario where FMBBR gradually enters the STARTUP phase. In this case, the behavior is much more user-friendly.

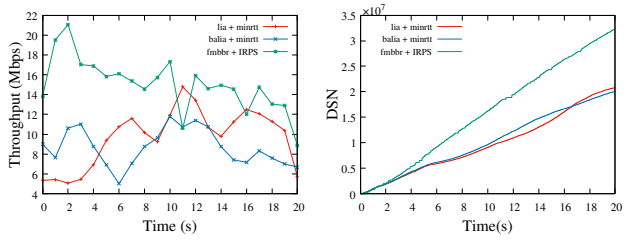


(a): STARTUP Together (b): STARTUP Sequential

Fig. 4: STARTUP Comparison

3) DSN and Throughput: Fig. 5(a) compares the throughput performance of BALIA and LIA using the default minRTT scheduler with FMBBR using IRPS under the same link conditions. The results show that FMBBR achieves higher throughput than the other two combinations. Compared to the more conservative MPTCP congestion algorithms, BALIA and LIA, FMBBR makes better use of available bandwidth and delivers superior overall throughput.

Fig. 5(b) shows the amount of data received over time during the same transmission period. As observed, FMBBR is comparable to BALIA and LIA between 0 and 2 seconds. This delay occurs because FMBBR is in the STARTUP phase, during which it explores and detects the available bandwidth. However, after this initial phase, FMBBR surpasses both BALIA and LIA for the rest of the simulation, achieving higher data transmission performance.



(a): Throughput (b): Receiving DSN

Fig. 5: DSN and Throughput

V. CONCLUSION AND FUTURE WORK

To ensure MPTCP fairness while fully utilizing the bandwidth of each subflow, we propose a user-friendly shared bottleneck detection and congestion control algorithm, FMBBR, which is based on SDN and BBR. FMBBR uses SDN to accurately identify shared bottleneck information for each MPTCP subflow. By adopting a step-by-step startup mechanism, subflows are sequentially brought into the BBR STARTUP phase. This approach not only increases the bandwidth detected by subflows during the initial phase but also reduces their impact on other network users. After detection, bandwidth is allocated to the subflows on the shared bottleneck link according to

their RTT values. Simulation results demonstrate that FMBBR achieves higher initial detection bandwidth compared to both standard BBR and non-step-by-step startup methods. At the same time, it minimizes the negative impact on other users' network quality and delivers superior performance compared to existing MPTCP congestion control algorithms.

The experiments are currently perform on simple local networks, in the future, the work will expand to more complex SDN-WAN networks. We also aim to conduct evaluations in real-world scenarios to validate its effectiveness and robustness under practical network conditions.

REFERENCES

[1] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "Tcp extensions for multipath operation with multiple addresses," Tech. Rep., 2013.

[2] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath {TCP}," in *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, 2011.

[3] C. Raiciu, M. Handley, and D. Wischik, "Coupled congestion control for multipath transport protocols," Tech. Rep., 2011.

[4] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath tcp: Analysis, design, and implementation," *IEEE/ACM Transactions on networking*, vol. 24, no. 1, pp. 596–609, 2014.

[5] R. Khalili, N. Gast, M. Popovic, and J.-Y. Le Boudec, "Mptcp is not pareto-optimal: Performance issues and a possible solution," *IEEE/ACM Transactions On Networking*, vol. 21, no. 5, pp. 1651–1665, 2013.

[6] T. Gilad, N. Rozen-Schiff, P. B. Godfrey, C. Raiciu, and M. Schapira, "Mpcc: Online learning multipath transport," in *Proceedings of the 16th International Conference on Emerging Networking EXperiments and Technologies*, 2020, pp. 121–135.

[7] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "Bbr: Congestion-based congestion control: Measuring bottleneck bandwidth and round-trip propagation time," *Queue*, vol. 14, no. 5, pp. 20–53, 2016.

[8] I. Mahmud, T. Lubna, Y.-J. Song, and Y.-Z. Cho, "Coupled multipath bbr (c-mpbbr): A efficient congestion control algorithm for multipath tcp," *IEEE Access*, vol. 8, pp. 165 497–165 511, 2020.

[9] W. Wei, K. Xue, J. Han, Y. Xing, D. S. L. Wei, and P. Hong, "Bbr-based congestion control and packet scheduling for bottleneck fairness considered multipath tcp in heterogeneous wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 70, no. 1, pp. 914–927, 2021.

[10] J. Han, K. Xue, Y. Xing, J. Li, W. Wei, D. S. L. Wei, and G. Xue, "Leveraging coupled bbr and adaptive packet scheduling to boost mptcp," *IEEE Transactions on Wireless Communications*, vol. 20, no. 11, pp. 7555–7567, 2021.

[11] S. Ferlin, Ö. Alay, T. Dreibholz, D. A. Hayes, and M. Welzl, "Revisiting congestion control for multipath tcp with shared bottleneck detection," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, 2016, pp. 1–9.

[12] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2014.

[13] Y. Liu, X. Qin, T. Zhu, X. Chen, and G. Wei, "Improve mptcp with sdn: From the perspective of resource pooling," *Journal of Network and Computer Applications*, vol. 141, pp. 73–85, 2019.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM computer communication review*, vol. 38, no. 2, pp. 69–74, 2008.

[15] D. Wischik, M. Handley, and M. B. Braun, "The resource pooling principle," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 5, pp. 47–52, 2008.

[16] D. Hayes, S. Ferlin, M. Welzl, and K. Hiorth, "Shared bottleneck detection for coupled congestion control for rtp media," Tech. Rep., 2018.

[17] J. Ye, R. Liu, Z. Xie, L. Feng, and S. Liu, "Emptcp: An ecn based approach to detect shared bottleneck in mptcp," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, 2019, pp. 1–10.

[18] R. F. Serfozo, "An equivalence between continuous and discrete time markov decision processes," *Operations Research*, vol. 27, no. 3, pp. 616–620, 1979.

[19] K. Nguyen, M. G. Kibria, K. Ishizu, F. Kojima, and H. Sekiya, "An evaluation of multipath tcp in lossy environment," in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2019, pp. 573–577.

[20] M. Hock, R. Bless, and M. Zitterbart, "Experimental evaluation of bbr congestion control," in *2017 IEEE 25th international conference on network protocols (ICNP)*. IEEE, 2017, pp. 1–10.

[21] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[22] C. Fernandez and J. L. Munoz, "Software defined networking (sdn) with openflow 1.3, open vswitch and ryu," *UPC Telematics Department*, 2015.