A Survey on Congestion Control and Scheduling for Multipath TCP: Machine Learning vs Classical Approaches

Maisha Maliha
School of Computer Science
University of Oklahoma
Norman, Oklahoma, USA
Email: maisha.maliha-1@ou.edu

Golnaz Habibi
School of Computer Science
University of Oklahoma
Norman, Oklahoma, USA
Email: golnaz@ou.edu

Mohammed Atiquzzaman
School of Computer Science
University of Oklahoma
Norman, Oklahoma, USA
Email: atiq@ou.edu

Abstract—**Multipath TCP (MPTCP) has been widely used as an efficient way for communication in many applications. Data centers, smartphones, and network operators use MPTCP to balance the traffic in a network efficiently. MPTCP is an extension of TCP (Transmission Control Protocol), which provides multiple paths, leading to higher throughput and low latency. Although MPTCP has shown better performance than TCP in many applications, it has its own challenges. The network can become congested due to heavy traffic in the multiple paths (subflows) if the subflow rates are not determined correctly. Moreover, communication latency can occur if the packets are not scheduled correctly between the subflows. This paper reviews techniques to solve the above-mentioned problems based on two main approaches; non data-driven (classical) and data-driven (Machine Learning) approaches. This paper compares these two approaches and highlights their strengths and weaknesses with a view to motivating future researchers in this exciting area of machine learning for communications. This paper also provides details on the simulation of MPTCP and its implementations in real environments.**

Index Terms—**Multipath TCP, congestion control, scheduling, deep reinforcement learning, machine learning**

I. INTRODUCTION

COMMUNICATION is key in many domains, such as defense, hospitality, technology, and space. In telecommunications, packet switching is a method of grouping data in smaller packets for faster communication [1]. One of the earliest packet-switched networks started with the Advanced Research Projects Agency Network (ARPANET) [2] in the United States, which is also called the forerunner of the Internet. Today, the Internet has expanded and now consists of a set of protocols for global communications. Basically, a protocol is a set of rules that the sender and receiver must agree on to communicate with each other. The two well-known Transport layer protocols are User Datagram Protocol (UDP) [3] and Transmission Control Protocol (TCP). UDP does not need any handshaking, which means the receiver does not send any acknowledgment to the sender when it receives a message. UDP thus leads to faster communication.

TCP [4] provides more consistent communication by considering handshaking between the sender and receiver. With millions of devices connected to the Internet, there is a demand for faster communication, but TCP fails to meet that need. It's because of TCP's congestion control algorithm, which decreases the throughput (message delivery rate) in response to the loss of packets in the network. Also, the handshaking of TCP increases the time necessary for a packet to travel, resulting in higher latency. Keeping in mind these problems, Multipath TCP (MPTCP) has been introduced by the Internet Engineering Task Force (IETF) [5] to use multiple paths effectively and efficiently between the sender and receiver.

TCP connections can experience packet losses or connection drops, resulting in a poor user experience [6]. MPTCP can use multiple TCP connections, known as subflows, in parallel to overcome TCP's limitations. One of the main goals of MPTCP is to control congestion and maintain traffic flows. Another focus of MPTCP is scheduling the packets over different subflows to send packets with the smallest round-trip time (RTT) [7], which is the time taken to send a data packet to the destination and receive an acknowledgment from the receiver. There are a set of traditional techniques such as Dynamic-Window Coupling (DWC) [8], Opportunistic Linked Increases Algorithm (OLIA) [9], Balanced linked adaptation (BALIA) [10], and Adaptive and Efficient Packet Scheduler (AEPS) [11] that control congestion or simultaneously schedule packets over multiple paths in MPTCP.

Since the standardization of MPTCP, a lot of classical approaches have been proposed to improve the performance of the network in terms of throughput and latency, but most of them perform poorly in highly dynamic networks. Recently, *data-driven* approaches, which are mostly based on Deep Reinforcement Learning, perform much better in dynamic networks because of their ability to learn the network conditions. To the best of our knowledge, there have been only a few proposals on controlling congestion while scheduling packets using deep reinforcement learning-based approaches. Although some methods have been proposed to control congestion, those works have not focused on reducing RTT. Some researchers have proposed schedulers to reduce latency, but they did not consider achieving high throughput.

Invited contribution

One of the biggest benefits of using MPTCP is its capacity to use all the available subflows and boost the network's throughput. However, to achieve high goodput, a scheduling strategy is important. Scheduling in MPTCP distributes packets over different subflows based on the smallest RTT. Schedulers using classical approaches, like [12]–[14], increased the throughput but failed to adapt to the dynamic nature of a real-world network. They were tested using simulation, which, of course, does not emulate the real-world scenario. Machine Learning and Reinforcement Learning models have improved the above drawbacks as they can learn from past experience and they are more robust to the dynamic nature of real-world networks; However, machine learning techniques are usually slower than classical approaches and need a huge dataset to train the models [15]–[17].

A. Contributions

The *objective* of this paper is to provide a brief overview of the existing work on MPTCP, including both classical and machine learning-based approaches. We discuss how previous researchers have addressed MPTCP challenges and summarized their solutions. Previous works have reviewed the existing congestion control and scheduling techniques for MPTCP [7], [18]–[21]. Among those works, some review papers have focused on either congestion control of MPTCP or only the scheduling of MPTCP, while others have reviewed only the existing work on MPTCP establishment. There are also some works that have mentioned both congestion control and scheduling but have not focused much on the scheduling-based works of MPTCP. The *contributions* of this paper are as follows:

- Discuss the difference between the traditional TCP and MPTCP communication protocols.
- Discuss in detail both the congestion control and the packet scheduling problems separately.
- Comparison between the performance of ML-based and classical algorithms in terms of controlling congestion and packet scheduling.
- Summarize basic concepts in MPTCP, including the establishment of MPTCP in real-world platforms and simulators.
- Highlights advantages and limitations of previous works that can help the readers investigate future improvements on MPTCP.

The rest of our paper is organized as follows; Section II describes the terminologies in communication and deep reinforcement learning. Section III compares TCP and MPTCP. Sections IV and V describe previous works on congestion control and scheduling of MPTCP, respectively. Section VI focuses on the performance of both congestion control and packet scheduling. MPTCP implementations in the kernel and NS-3 are described in Section VII. Lastly, in Section VIII, we conclude our survey by discussing future works in MPTCP congestion control and scheduling.

II. BACKGROUND & TERMINOLOGIES

A. Overview of TCP

TCP is a connection-oriented communication standard that computer applications use to communicate over a network. It is a packet transfer protocol in the Transport Layer [22] of the TCP model. TCP uses only one dedicated path for packet transfer. Though TCP guarantees data integrity of the packets, it has to face packet loss, delay and other problems which are discussed in Section III. Also, network congestion is another major problem in TCP which is discussed briefly later. Section II-B summarizes some concepts in TCP which would be in common with MPTCP and they are also used in congestion control.
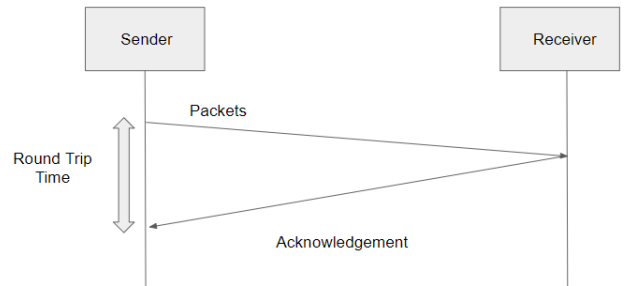

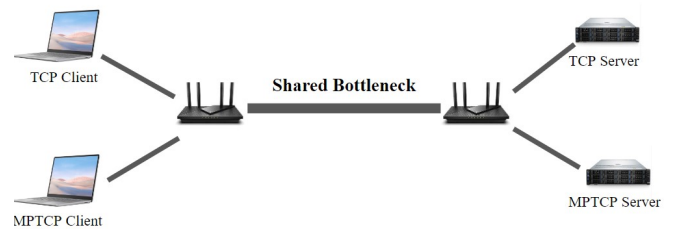Fig. 1. Illustration of the RTT of a packet from the sender to the receiver.

Fig. 2. Illustration of shared bottleneck scenario in a network [23].

B. Some Concepts in TCP

- **Round Trip Time (RTT):** The time required to send a packet from the client to the server, and the time it takes for the server to receive an acknowledgment about receiving the packet is known as the round trip time (RTT). Reducing the round trip time is a primary focus of MPTCP. Figure 1 illustrates the meaning of RTT.
- **Throughput vs Goodput:** Throughput refers to the total number of packets transferred to the destination within a fixed time frame. On the other hand, Goodput is the number of meaningful packets that are delivered to the destination within a given time frame.
- **Low Latency vs High Latency:** Latency refers to the amount of time required to send a packet from source to destination and back again. Low latency is always preferable.

- **Congestion Window (CWND):** The congestion window decides the number of bytes or how many packets will be sent at a given time. Depending on the larger congestion window size, the throughput will also be maximized. Congestion window size has been determined by the slow start and congestion avoidance phases of TCP which will be discussed in the next part.
- **Bottleneck vs Shared Bottleneck:** Bottleneck occurs when there is not enough network capacity in a connection to handle the current volume of traffic. On the other hand, when a bottleneck link is shared between multiple subflows, it is referred as a shared bottleneck which is useful for maximizing the throughput. Figure 2 illustrates the concept of a shared bottleneck scenario.

C. Congestion Control in TCP

TCP's congestion control mechanism has three phases; (1) slow start phase; (2) congestion avoidance phase; and (3) congestion detection phase. The basic difference between these three phases is the rate of increase in the congestion window size.

- **Slow Start Phase:** Slow start phase works as a part of the congestion control algorithm in TCP by controlling the amount of data flow in a network. When a network becomes congested from excessive data in the network, the slow start phase chokes the traffic by limiting the congestion window size. In the slow start phase, the sender sends a packet that contains its initial congestion window, and the client responds with its maximum buffer size after receiving the packet. If the sender gets the acknowledgment from the receiver, the number of packets to be sent to the receiver is doubled. This procedure continues until no acknowledgment is received. The acknowledgment may not be received for two reasons: if congestion occurs or the window limit of the client is reached.
- **Congestion Avoidance Phase- Additive Increase:** The congestion avoidance phase starts when the congestion window size of the TCP reaches a threshold in the slow start phase. In this phase, the size of the congestion window increases linearly. To elaborate, assume the congestion window size at time t is 20 and all the packets have been transmitted successfully, then the congestion window size at time $t + 1$ will be 21.
- **Congestion Detection Phase- Multiplicative Decrease:** If congestion occurs in the slow start phase or congestion avoidance phase, the congestion window size is decreased. This is called the multiplicative decrease phase, where TCP follows an exponential reduction of the congestion window. The additive increase and multiplicative decrease phases of the congestion avoidance and detection phases are referred to as Additive Increase Multiplicative Decrease (AIMD). An example of AIMD is shown in Figure 3.
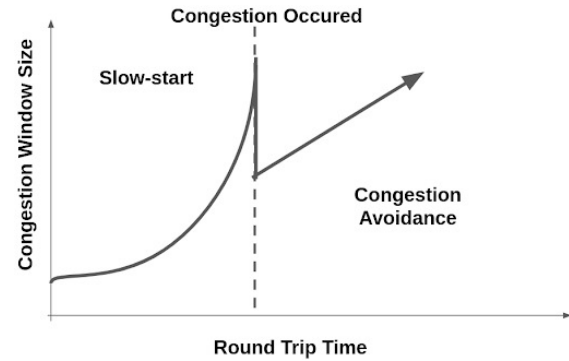

Fig. 3. Change in the congestion window in AIMD algorithm when a packet loss is encountered.

D. Overview of MPTCP

As opposed to TCP, which solely considers one path to transfer data, MPTCP is a transport layer protocol that allows the transfer of packets along multiple paths between the sender and the receiver. This helps the network increase its load capacity, thereby transferring a larger number of packets compared to TCP. The paths in MPTCP are called subflows. When one or more of the subflows fails to send a packet, it can flow through other subflows, leading to a fault-tolerant network. MPTCP is used in several areas of communication where there is a need for high throughput and very low latency during packet transfer. MPTCP is used in different applications such as online streaming, networking, gaming industries, VPNs. Figure 4 depicts the use of MPTCP where a cellphone may use either one of the subflows from two subflows to get connected to the server: one is the Wifi, and another subflow is the 5G network. The following section explains the procedure for establishing the MPTCP communication.
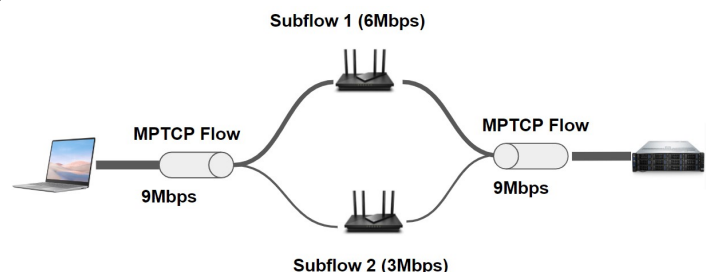

Fig. 4. Overview of MPTCP

.

1) Establishment of MPTCP Connection: The establishment of MPTCP between a sender and the receiver has two stages; In the first step a single flow is established. This phase is similar to TCP. Then, subsequent subflows are created. In the first stage, the sender and the receiver use one subflow to set up the MPTCP connection between them by sharing randomly generated keys. This lays the foundation for creating further paths between the sender and the receiver. Figure 5 shows the establishment of an MPTCP connection using all subflows.
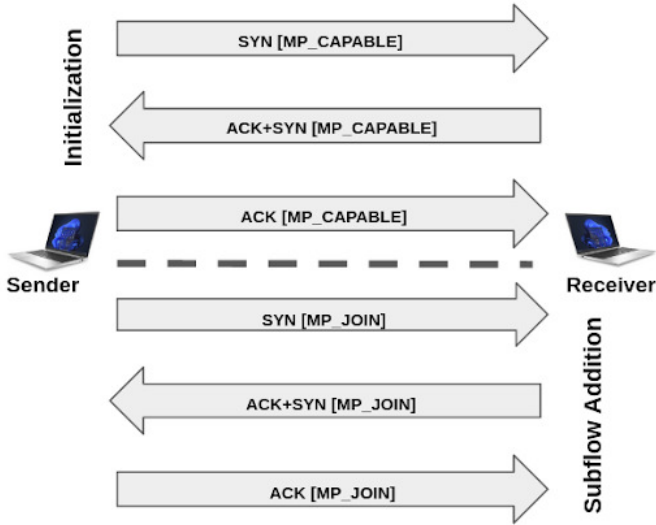
Fig. 5. Establishment of an MPTCP connection.

keep the necessary information and forget the unnecessary ones. Figure 6 shows the different components of an LSTM and compares with RNN. The LSTM-based framework is very popular to create Deep Reinforcement Learning-based congestion control system for MPTCP [26], [27].
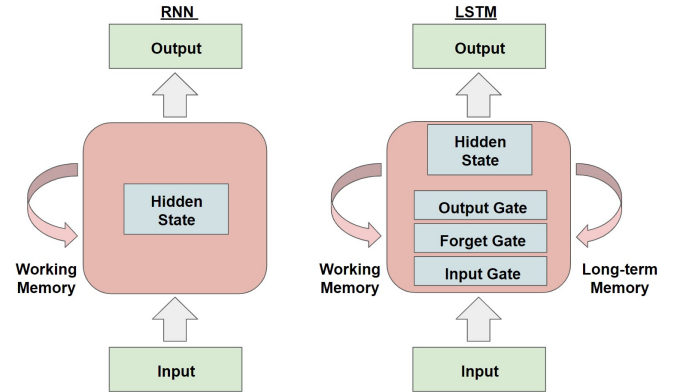

Fig. 6. Architecture of RNN (left) vs LSTM (right).

In MPTCP, after the establishment of the initial handshake as in TCP, the subsequent subflows are also handshaked [24]. MPTCP follows three-way handshaking consisting of SYN (synchronize), ACK+SYN and ACK (acknowledge). In the SYN packet, the sender shares its own token and a random nonce (number). Here the token is the hash value of the key using some cryptographic function which can be calculated in the initial phase with the keys exchanged in that phase. Subsequently, in the SYN+ACK packet, the receiver creates an HMAC (Hash-based message authentication code), the receiver's token and its nonce.

The option MP_CAPABLE is used to check whether the remote host is MPTCP enabled in the initial subflow. The option MP_JOIN is used in the additional subflow establishment to associate with MPTCP connections. Lastly, the sender responds with its HMAC in the ACK packet. [24]

E. Overview of Machine Learning Concepts used in Congestion Control in MPTCP

1) RNN and LSTM: RNN (Recurrent Neural Network) is one kind of neural network that passes the output from the previous steps to the next steps. RNN consists of the input layer, hidden layers, and the output layer. Its hidden state remembers the previous information to predict the next output. RNN works well in terms of correlation, while LSTM (Long Short-Term Memory) not only connects the correlation but also focuses on the context of the information. LSTM is an artificial neural network that follows feedback connections and stores previous information to predict the next one. It is a modified version of the RNN that solves the vanishing gradient problem [25] and can easily process longer sequences. LSTM has an input gate, an output gate and a forget gate. The Input Gate takes an input and vectorizes the input value. The forget Gate is responsible for forgetting unnecessary information, while the output gate generates the output. This helps the framework

2) Reinforcement Learning and Deep Q-Learning (DQN): A Markov model [28] consists of a tuple $(s, a, r(s, a))$, where s is the current state vector, a is the vector of actions an agent takes, and R is the reward or the feedback the environment provides for the agent, given the current state and action. In a Markov Decision Process, the agent makes a set of actions, called policy, to maximize its expected reward. The function $Q(s, a)$ defines the optimal (*i.e.*, maximum) expected reward the agent can get, given the current state s and action a. Evaluating Q is important to plan for the optimal policy. Due to the stochastic nature of many environments and agents, calculating the exact value of Q is usually not possible. Instead, the agent learns Q values from its experience; this is called Q-learning, a type of reinforcement learning.

DQN Reinforcement Learning (DRL) uses a deep neural network (DNN) to learn/estimate the Q function. The deep neural network could be RNN, LSTM, CNN etc. In contrast, a traditional Q estimation (e.g., Temporal Difference (TD) learning [29]), is usually a memory table that stores all the previous records of the different steps that have been taken, along with their rewards which is not scalable for an environment with large size of space and actions, or when the action/state space is continuous. Deep-RL has been used in communication for learning the communication strategy between multiple agents. DRL is also used in MPTCP implementation to control congestion and schedule packets to maximize throughput and get low latency. In MPTCP communication network, the state can be throughput, sending rate, RTT, and loss of packets; actions should be increasing/decreasing congestion window size and reward is the measurement of either good or bad performance of the network, based on the state and action.

3) Actor-critic model: An actor-critic has two major components: an actor and a critic. An actor takes the state of the current environment and determines the best action that needs to be taken, depending on that state. Whereas the critic works

for the evaluation role by taking the environment state with actions and returning a score/reward (*e.g.,* Q) to decide how good is that action for that state. Determining the best action depends on the Q score, which is calculated separately in the critic network. An actor-critic learning technique learns both a policy function and a value function at the same time. The value function aids in enhancing the value function's training procedures, while the policy function instructs on how to make decisions.

4) Transformers and Self-Attention: The transformer is an encoder-decoder model and has been used in many applications [30]. Self attention follows the encoder part of transformers and can be used in many communication applications such as congestion control [27], or scheduling packets for MPTCP. In natural language processing or machine translation, self-attention for a particular word measures the dependency of that word on the other words; more relevant words have a higher value in self-attention, and loosely connected words have lower values [31], [32]. This concept is brought into the area of communication and subflows. Here the words have been replaced by the subflows. In MPTCP, self-attention checks the dependency of a subflow with the other subflows by assigning different weights to the states of the other subflows. Here the states of the subflows may refer as RTT, packet loss, packet delay, and throughput [27].

III. TRADITIONAL TCP VS MULTIPATH TCP

In this section, we discuss some major challenges in network communication, how MPTCP and TCP address them in different situations, including the comparison of MPTCP and TCP performance.

A. Packet Loss

Packet loss happens in a network when a packet fails to reach the receiver. Packet loss in a network indicates network congestion, disruption, or even a complete loss of connection. When a TCP connection encounters a packet loss, it considers a sign of network congestion and, therefore, halves the size of the congestion window and the threshold value; hence the throughput decreases. However, MPTCP is more robust in such cases; whenever it sees one of the subflows having packet loss, it reduces the congestion window of that subflow, while the other subflows remain intact. This was experimentally tested [33] where TCP and MPTCP transmission performances were compared using WiFi and LTE. In this real-world experiment, 750 individuals from 16 nations utilized a crowd-sourced smartphone application for 180 days. Deng et al. [33] compared MPTCP and TCP performance via WiFi or LTE from 20 different places across seven cities in the United States. The fastest link of normal TCP exceeds MPTCP performance for short flows. In this scenario, MPTCP fails to select an appropriate communication path to reduce transmission time for a small amount of data; hence, the packet loss increases for MPTCP. However, TCP experiences greater packet loss than MPTCP in log-distance communication experiments.

B. Packet Delay

Packet delay in a network is the time taken to transfer a packet of data from sender to receiver. The packet delay is highly affected by the number of routers or switches in the route, the nature of the path the packet follows, and the congestion in that path. The vulnerability of packet delay in both TCP and MPTCP scenarios can be described by one communication example. If a computer is connected via both wireless and wired connections, it may communicate with servers using TCP or MPTCP. In terms of TCP, if the connection is established over the wireless connection, it will have a greater overall delay than MPTCP. Wireless connection experiences high packet loss as opposed to the wired connection which leads to higher latency [34].

MPTCP has the choice to choose either a wired or wireless connection, it may transfer the packets via the wired connection if it experiences major packet delay through a wireless connection. Raiciu et al. [35] have compared the performance of MPTCP and TCP-based on the packet delays in different network topologies. Their findings reveal that MPTCP can achieve 90 percent bandwidth utilization and low overall packet delay when the number of subflows is two and eight in the VL2 [36] and FatTree network [37] topologies, respectively. The result also showed that MPTCP not only increases the bandwidth but also increases the robustness to network changes by lowering the packet delay.

C. Out-of-Order Packets

In TCP, a message is divided into multiple parts, known as packets. Each of these packets is given a unique number known as the sequence number. When these packets reach the receiver, it uses the sequence number to put them in order to retrieve the message. If these orders are not maintained during transmission, the delivery is called out-of-order delivery of packets. UDP is notorious for these deliveries as it does not consider any handshaking when each packet is received. But in TCP, the next packet is not sent until it gets the acknowledgment from the previous packet. If it gets a time out or negative acknowledgement (NACK) [38], the packets are sent again; therefore, out-of-order delivery is rare in TCP. However, in MPTCP, there is a high chance of out-of-order delivery of packets as they use different subflows with different delays. Therefore, scheduling is one of the most challenging tasks in MPTCP, and a lot of work has been done to address this issue. Yang et al. [39] have mentioned a situation where the jitter can happen for transferring data in MPTCP. They tackled this issue using an innovative traditional scheduling process named Delay Aware Packet Scheduling technique to remove the jitter in the packets. Han et al. [17] used a queue to keep redundant packets that may get lost.

D. Round Trip Time

Round Trip Time (RTT) has been discussed in section II. Chen et al. [40] have compared the performances of TCP and MPTCP over WiFi and cellular networks where the authors

compared the RTTs of the transmission protocols. They conducted two sets of experiments; in the first experiment, they used small-sized files, and in the second, large-sized files were used. In the first experiment, the file size varied from 8KB to 32MB. When WiFi is the default route, there is no discernible gain in MPTCP download performance over TCP. For small file downloads (such as 64 KB), the single route via WiFi delivers the optimum speed. However, a single LTE (Long-Term Evolution) [1] channel becomes the optimum option for relatively longer traffic flows. MPTCP outperforms TCP for larger files.

IV. CONGESTION CONTROL OF MPTCP

Congestion control is a concept of controlling congestion in a network and could happen in both TCP and MPTCP. Congestion occurs when there is too much data that needs to be sent through a network. Congestion control regulates the flow of data packets into the network, allowing for efficient use of a shared network infrastructure and preventing congestion collapse. In TCP, where there is only one subflow, the network is easily congested.

Different types of algorithms have been proposed to improve the congestion in TCP networks, such as TCP Cubic [41], TCP Vegas [42], and TCP Reno [42]. MPTCP provides several subflows, which results in a reduction of congestion. MPTCP has been designed to address the congestion issue, while still having the traffic flowing like a single-path TCP. A naive implementation of CC in a multipath setting would be using regular TCP congestion control for each subflow; however, it is not efficient as MPTCP which uses multiple concurrent TCP connections. Having a congestion control that manages the packet flows on subflows concurrently seems more efficient. For this purpose, many methods have been proposed to improve congestion in MPTCP. Congestion control algorithms for MPTCP are classified as classical and machine learning approaches which will be discussed in the following subsections.

A. Classical Congestion Control Approaches

Most of the existing congestion control algorithms in MPTCP setting focus on the Congestion-Avoidance (CA) phase that solely considers long flow transmissions and does not focus much on the slow start phase. The congestion avoidance phase prevents a network from being overflooded by data such that it discards packets with low priority to be delivered, and the rate of transmission rises linearly over time. Another approach is to focus on Slow Start. The Slow Start phase limits the quantity of data to be sent over a network to avoid congestion. However, it causes exponential growth of the congestion window in the uncoupled Slow-Start (SS) phase, leading to buffer overflow from burst data. In terms of solving the mentioned problems, Yang et al. [43] have proposed a Throughput Consistency Congestion Control (TCCC) algorithm which consists of both Coupled Slow-Start (CSS) and Aggressive Congestion Avoidance (ACA).

The usage of CSS prevents packet loss brought on by large data bursts and ACA works on getting fair bandwidth which is shared in congestion avoidance. Their proposed framework enhances transmission efficiency. However, the CSS algorithm only plays a part in the initial slow start phase of MPTCP. As the subflows of MPTCP belong to different phases in congestion control (see Section II-C), the CSS algorithm needs much extra consideration, which makes congestion control more challenging in MPTCP [43].

Traditional AIMD used in TCP shows poor performance adaptation in terms of network state-changing situations in MPTCP. Gilad et al. [44], have presented a method named MPCC that uses online learning. Their implementation has been performed in Linux kernel and the method has been tested on different network conditions and many different network topologies. In terms of improving the implementation, their analysis needs to be reached beyond parallel link networks. As further research, they have mentioned about boosting the performance for short flows and solving the bandwidth mismatch problems on network paths.

An energy-aware based congestion control algorithm (ecMTCP) has been developed by Le et al. [12] where the method distributes traffic between the most crowded and least crowded paths, as well as across paths with different energy costs, to achieve load balancing and energy savings. For simulation purposes, they used NS-2 simulator [45], and their design mechanisms can work on getting higher throughput in terms of both TCP and MPTCP flows. The main goal was to shift the traffic to less energy-intensive and less crowded paths. Cao et al. [13] proposed weighted Vegas (wVegas), a delay-based congestion control scheme for MPTCP. This algorithm has detected the packet queuing delay of each path and ultimately has decreased the packet load of congested subflows by increasing the load of the less congested one. This framework performed traffic shifting which can cause less packet losses and provide better traffic balance in subflows. Cao et al. used NS-3 simulator [46] to conduct the simulation and build a Network Utility Maximization Model by proposing an approximate iterative algorithm to reach their aim of controlling congestion.

Ji et. al [14] mentioned that existing multipath congestion control algorithms are unable to quickly adjust to dynamic traffic due to the heterogeneous Quality of Service (QoS). QoS refers the technologies that work on a network to manage traffic and enhance performance by reducing packet loss, delay, and latency in a network. It may lead to poor performance in certain network environments. To mitigate these issues, firstly, the authors have noticed the performance constraints of the most recent multipath congestion control algorithms through vast experimentation. Then, they used a unique control policy optimization phase referred to an adaptive QoS-aware multipath congestion management system that can quickly adapt to network changes. Their method uses the Random Forest Regressing (RFR) [47] method to carry out QoS-specific utility function optimization to adapt and encourage the improvement of the selected performance metric. They

---

[1]wireless broadband communication standard came before 4G network

conducted the implementation in Linux kernel and showed their work outperformed most of the multipath congestion control methods such as wVegas [13], Opportunistic Linked Increases Algorithm (OLIA) [9], Balanced Linked Adaptation (BALIA) [10].

Singh et al. [48] improved the Opportunistic Linked Increases Algorithm [9] and Dynamic-Window Coupling (DWC) [8]. The authors provided a mechanism to reduce the overall packet reordering delay and focused on the buffer size in the receiver side. Their proposed work showed good performance in terms of various bottleneck scenarios.

Hassayoun et al. [8] proposed a multipath congestion control scheme called Dynamic-Window Coupling (DWC) to obtain higher throughput to each end-to-end multiple paths. The authors also detected shared bottlenecks by monitoring loss and delay signals, and then grouped their congestion control mechanism over all subflows that shared a common bottleneck. Detecting the bottleneck for network conditions and regrouping the subflows in terms of the same bottlenecks leads to higher throughput. They also introduced subflow sets, a concept for enabling subflows to smoothly switch between independent and shared bottleneck-based congestion control. The algorithm has been implemented in NS-2 simulator. As future research, the authors introduced the possibility of including "memory" into the detection method for detecting the previous subflow groupings.

Ferlin et al. [49] have mentioned that increasing the bandwidth of multiple links and getting higher throughput will be impossible if two or more paths do not share a bottleneck. They found out the nonshared bottleneck paths of the coupled congestion control for links, they referred to it as a penalty, and to overcome it, they implemented shared bottleneck detection (SBD) algorithm for MPTCP. This work can balance congestion and throughput. Their observation has shown that in the case of non-shared bottleneck scenario, the maximum throughput can be achieved up to 40% with two subflows. Also, the throughput gain increased by above 100% when the number of subflows increased to five. Their implementation has been performed in Linux kernel and for emulation purposes, they have also used CORE network emulator [50].

B. Machine Learning Approaches for Congestion Control in MPTCP

Though lots of work have been done in classical-based approaches for congestion control of MPTCP. But for controlling congestion, classical-based approaches focus solely on different types of congestion indicators (*i.e.*, packet loss or RTT). In the case of classical-based approaches, the decision-making process totally depends on these unpredictable factors, which leads to poor performance. Whereas ML-based approaches aim to provide decisions based on experience, and can adapt well to any network situation. Thus, ML-based approaches outperform classical-based methods [51].

In reality, networks are dynamic, and the state of the network changes frequently. Due to that, MPTCP performs poorly in many practical situations as MPTCP has to adapt in new network states. Zhuang et al. [52], introduced a Reinforcement Learning technology that can learn the best route to send TCP packets such that the throughput has been maximized. They proposed a simple algorithm for controlling multipath congestion, where congestion control has been approached as a multi-armed bandit [53] issue based on online learning (MP-OL), which allowed flexible and adaptive transmission rate adjustments for each subflow with good performance.

In [26], the authors proposed a Deep Reinforcement Learning (DRL)-based framework to control congestion where a single DRL agent has been utilized to perform congestion control for all MPTCP flows to maximize the total utility. Figure 7 illustrates the concept of DRL for MPTCP congestion control. They implemented the MPTCP in the Linux kernel and used an LSTM-based neural network under a DRL framework to develop a representation for all active flows. Their work was the first work where the authors incorporated the LSTM-based representation network into an actor-critic architecture for controlling congestion which used the deterministic policy gradient [54] to train the critic, actor, and LSTM networks.

He et al. [27] worked on increasing/decreasing the sending rates of packets in response to congestion, where each DRL agent can control the congestion window size of each subflow. Their proposed DRL-based MPTCP framework also included self-attention, which has been used to check the dependencies of one subflow with the weighted sum of other subflows. They compared their work with DRL-CC [26] and showed their method outperforms DRL-CC.
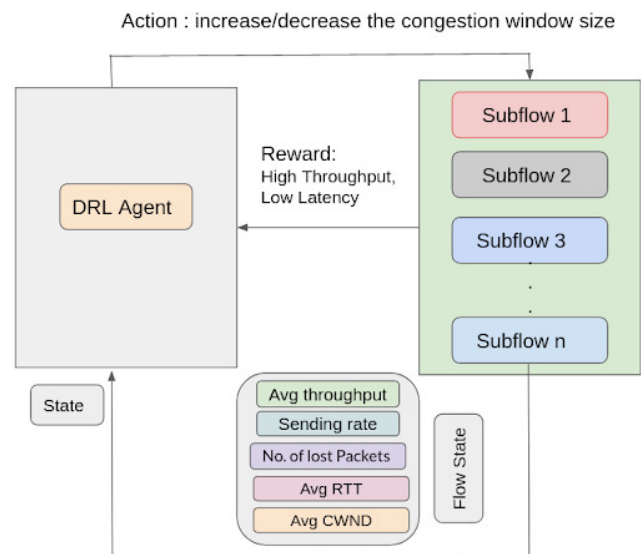

Fig. 7. Subflows are controlled by a DRL agent.

Li et al. [55] proposed a method called SmartCC which can learn a set of congestion rules for observing the environments and taking actions to adjust the congestion window size of each subflow. For the MPTCP implementation task, the authors used the NS-3 simulator.

The Internet of Deep Space Things, or IoDST, offered

communication services for mission spacecraft that send video data. To improve TCP throughput and stream playback, Ha et al. [15] designed a congestion control framework for MPTCP, which can be used for data streaming transmission. Their proposed Q-learning and Deep Q-Network (DQN)-based congestion control scheme calculated the ideal congestion window for data transfer in IoDST conversations.

Xu et al. [56] proposed the SGIN-based High-Speed Railway (HSR) scenario with MPTCP. Space-ground integrated networks (SGINs) has been referred to as promising network architecture that provides seamless, high-rate, and reliable data transmission with incredibly wide coverage. By utilizing MPTCP in the SGIN, simultaneous data transfer over terrestrial and satellite networks has been made possible. However, due to MPTCP's current congestion control (CC) mechanisms, it's difficult to know the difference between negative effects (like packet loss and/or increased round-trip time) brought on by congestion and those brought on by handovers. This may lead to severe performance degradation in the SGIN-based HSR scenario, where handover may occur frequently. To solve it, a DRL-based novel approach has been proposed to improve the goodput which outperformed other state-of-the-art algorithms.

Xu et al. [57] presented a DRL-based novel framework for traffic engineering that can make decisions under the guidance of actor-critic networks. In their work, the state consisted of two components, such as the throughput and the delay of each communication session. On the other hand, the action has been defined as the solution to Traffic Engineering (TE) problems. The authors used the NS-3 simulator, and the reward of the model was the sum of the output from the utility function for an entire communication session. The utility function was a function of the throughput and delay of the network, which depicted how the network can perform. In the paper, each session had 20 iterations. In each iteration, the agent sent its actions to the environment and recorded the value from the utility function before updating the reward value. While they considered only one DRL agent (*i.e.*, decision maker) in their framework, adding multiple agents can be considered to further improve the performance.

Pokhrel et al. [58] introduced a transfer learning-based MPTCP framework for Industrial IoT, where the neighboring machines can collaborate to learn from each other. In their approach, when a new DRL system controlling the IoT network joins the environment, it can use the idea of transfer learning. [2] NS-3 was used to simulate the algorithm. Their model has been proven theoretically and needs further research to determine its performance in a real-world situation.

V. SCHEDULING OF MPTCP

Scheduling of MPTCP decides the amount of data that needs to be scheduled to different subflows based on getting the higher performance (high throughput, low latency, less packet loss) in MPTCP. In this section, different classical and ML-based approaches have been discussed which can be used to schedule packets in MPTCP.

A. Classical Approaches for Scheduling in MPTCP

Hwang et al. [59] dealt with the problem of scheduling small-length packets. However, the authors mentioned MPTCP is usually advantageous for long-lived flows, and it performs worse than single-path TCP when the flow size is tiny (*e.g.*, hundreds of KiloBytes). In this scenario, the quickest method is preferable since latency is far more critical than network bandwidth with such tiny data deliveries. The regular MPTCP packet scheduler may pick a slow path if the fast path's congestion window is unavailable, resulting in a delayed flow completion time.

To address this issue, Hwang et al. [59] suggested a novel MPTCP packet scheduler that momentarily blocks the slow path when the latency difference between the slow and fast paths is considerable, allowing the tiny quantity of data to be delivered swiftly via the fast path. The authors used the method to find the subflow with the lowest RTT regardless of the availability of the congestion window, and then they used the existing Lowest-RTT-First policy [60] to choose the optimal subflow. They then returned the best one if the difference between the best subflow RTT and the minimum RTT is less than a certain threshold. They picked 100ms for threshold delay when testing 3G and WiFi networks in this paper.

Chaturvedi et al. [11] analyzed different existing schedulers and identified some current outstanding concerns, such as head-of-line (HoL) blocking and out-of-order packet delivery. HoL blocking may occur when a single data packets queue may wait to be transmitted and the packet at the head of the line may not be able to move ahead due to congestion [61]. These problems reduce MPTCP performance and to mitigate the issues, the authors have presented an adaptive and efficient packet scheduler (AEPS). This novel MPTCP packet scheduler not only addresses these concerns but also offers high throughput with a short completion time by using the capacity of all available pathways. AEPS can send data packets to the receiver in the order they were received, and its performance is unaffected by the size of the receiver buffer, or the size of the data being transmitted. The AEPS has been developed with three objectives: (1) packets should arrive to the receiver buffer; (2) all pathways' bandwidth should be used; and (3) completion time should be as short as possible. According to the authors, the first condition assisted AEPS in resolving the HoL blocking and received window-limiting issues by sending packets to the receiver buffer in sequence. The second condition summed the bandwidth of each interface (path) by using all accessible pathways to the MPTCP source, which also helped to enhance throughput. The third criterion aided in choosing the routing for each packet so that the total network completion time can be minimized.

Dong et al. [62] thoroughly compared existing scheduling algorithms and guided the development of new scheduling algorithms in 5G. The authors examined the influence of several

---

[2]Transfer learning uses a previously trained model as the foundation for a new model on a different task.

network parameters, such as RTT, buffer size, and file size, on the performance of current extensively used scheduling algorithms over a wide range of network circumstances. The paper compares the Lowest-RTT-First [60], Delay-aware packet scheduler (DAPS) [63], Out-of-order transmission for in-order arrival scheduler (OTIAS) [64] and Blocking estimation-based MPTCP scheduler (BLEST) scheduling [65] algorithms. The number of timeouts and flow completion time are compared in the path heterogeneity test. The results showed Lowest-RTT-First has the most timeouts, while the BLEST has the fewest. BLEST surpasses other algorithms in varying buffer size outcomes, followed by OTIAS and DAPS. Since BLEST can dynamically predict whether head-of-line blocking will occur and hence minimizes the quantity of out-of-order packets. In the different file size tests, BLEST and LowRTT perform better than DAPS, and OTIAS outperforms BLEST.

Le et al. [66] tackled the problem of out-of-order delivery in MPTCP. Because of the diverse nature of latency and bandwidth on each channel, the out-of-order packet issue becomes severe for MPTCP. To solve this issue, the authors presented the forward-delay-based packet scheduling (FDPS) method for MPTCP. The technique is divided into two parts: predicting the forward delay differences across pathways and picking data to send through a path when the congestion window is available.

B. Machine Learning Approaches for scheduling in MPTCP

In recent times, many ML-based approaches have been proposed to improve the scheduling mechanism of MPTCP. Though classical approaches achieve good performance in terms of scheduling in MPTCP, ML-based approaches also show promising result and becomes popular in terms of getting higher throughput with lower latency than non-ML methods.

Wu et al. [16] applied a learning-based technique to schedule packets in the different paths of an MPTCP. The authors have presented FALCON, a learning-based multipath scheduler that can adapt to changing network circumstances quickly and correctly using meta-learning. The meta-learning algorithm comprises two parts: offline training and online training parts. The online learning module captures the network-changing conditions whereas the offline learning module takes the experience(data) from the online module and divides the experience into different groups depending on the network conditions.

Han et al. [17] used the technique of redundancy of packets to reduce packet loss by suggesting EdAR (Experience-driven Adaptive Redundant packet scheduler). In the face of dramatic network environment changes, EdAR enables dynamically scheduling redundant packets using an experience-driven learning-based strategy for multipath performance enhancement. To allow accurate learning and prediction, a Deep Reinforcement Learning (DRL) agent-based framework has been created that learns both the network environment and the optimal course of action. EdAR has two transmission modes: standard transmission and redundant transmission. Standard transmission follows the regular data transition. Regarding the

redundancy transmission, there is a buffer called redundant buffer. The redundant buffer holds packets that have already been transmitted but have yet to be acknowledged. If a new packet is transmitted from the send buffer on a subflow, it is copied to the redundant buffer. If a packet in the redundant buffer is not sent out or acknowledged, it is deleted from the redundant buffer. Silva et al. [67], used linear regression [68] to predict throughput and latency in MPTCP subflows, and proposed Artificial Neural Network [69]-based linear classifier to choose the best subflow which can provide better performance in MPTCP scheduler. They implemented their work in NS-3 simulator.

VI. CONGESTION CONTROL AND SCHEDULING OF MPTCP

Few works have been done focusing on congestion control and scheduling the packets of MPTCP at the same time. Those works get higher throughput and lower latency in terms of performance evaluation. Though further research regarding congestion control with packet scheduling is needed to be done. This section reviews classical and machine learning approaches that have been done in this domain.

A. Classical Approaches on both Congestion Control and Scheduling of MPTCP

Wei et al. [23] proposed a model that gets higher throughput when the networks do not go through a shared bottleneck. Their work had two outcomes: (1) When no congestion would occur, their method has been able to get higher throughput than a single TCP. (2) When there is congestion in the network, their method has at least the same throughput as TCP. Their method also measured how severe or minor the congestion is in the network. They have introduced both SB-CC (Shared Bottleneck-based Congestion Control Scheme) and SB-FPS (Shared Bottleneck-based Forward Prediction packet Scheduling scheme), where SB-CC can detect shared bottlenecks and estimate the congestion degree of all subflows. SB-FPS can perfectly schedule data in shared bottleneck and can also distribute data according to the congestion window size of each subflows. For implementing MPTCP, they used the Linux kernel and achieved higher throughput.

B. Machine Learning Approaches on both Congestion Control and Scheduling of MPTCP

Pokhrel et al. [71] have introduced the Deep Q learning (DQL)-based method to control congestion and schedule packets for MPTCP. Their proposed DQL framework has utilized the LSTM-based recurrent neural network where in their framework the Q function provided the logarithm value of goodput for the previous iteration. Here, the policy function was the actor-critic of two LSTMs and the value function was the reward. They considered RTT, throughput, and sending rate as the state. Depending on the state, their model provided action on whether window size needed to be increased or decreased and what changes can be taken in the schedule of packets for the subflows. In their work, the reward was the

TABLE I
PERFORMANCE MEASUREMENT OF REVIEWED PAPERS FOR CONGESTION CONTROL AND PACKET SCHEDULING IN MPTCP.

Approaches	Paper	Feature	Strength	Limitations	Implementation
ML	DeepCC [27]	Congestion Control	increasing/decreasing the sending rates of packets	increased computational time	Linux Kernel
	DRL-CC [26]	Congestion Control	high throughput	complexity by large state space	Linux Kernel
	SmartCC [55]	Congestion Control	dealt with multiple communication path in heterogeneous networks	did not consider TCP-friendliness issues	NS-3
	IoDST [15]	Congestion Control	calculated the ideal congestion window	did not use real experiments or emulated tests	computer simulations
	DRL for Handover-Aware MPTCP CC [56]	Congestion Control	higher goodput than other state-of-the-art algorithms	training time becomes longer	Linux Kernel
	MPTCP Meets Transfer Learning [58]	Congestion Control & Packet Scheduling	improved the efficiency of newly deployed machines	not proven in practical situations	NS-3
	FALCON [16]	Packet Scheduling	can adapt in network changing conditions	needs to understand the learning outcome	Multipath QUIC [70]
	EDAR [17]	Packet Scheduling	enabled dynamically scheduling redundant packets	increased computational time	NS-3
Classical	TCCC [43]	Congestion Control	improved efficiency in nonshared bottleneck scenario	needs extra consideration for different phases of MPTCP	NS-3 and Linux kernel
	MPCC [44]	Congestion Control	tested on different network conditions	bandwidth mismatch problems on network paths	Linux kernel
	ACCeSS [14]	Congestion Control	quickly adjust to dynamic traffic	performance can be improved	Linux kernel
	CC MPTCP with shared bottleneck detection [49]	Congestion Control	balancing congestion with improving the throughput	needs to improve it's robustness	Linux kernel and CORE
	Packet scheduling for multipath TCP [59]	Packet Scheduling	decreased the completion time of short flows	needs to improve overall transmission rate	Linux kernel
	AEPS [11]	Packet Scheduling	high throughput and low completion time	performed poorly in heterogeneous networks	Linux kernel and NS-3
	wVegas [13]	Congestion Control	less packet losses	needs to effectively handle multiple extended high-speed paths	NS-3
	DWC for MPTCP-CC [8]	Congestion Control	high throughput	did not mention their model performed better than others	NS-2

summation of all the Q functions for all subflows. Similar to other RL algorithms, the optimal decision was learned to maximize the reward. They have made their MPTCP implementation in the Linux kernel and achieved low delays with maximum goodput.

VII. IMPLEMENTATION OF MPTCP

In this section, we describe different ways of implementing MPTCP either in real hardware (kernel) or in simulator and list some of the works for each type of implementation as the reader's reference. Previously, NS-2 simulator has been used for implementing MPTCP [8], [72]. Now, most of the recent works have focused on implementing MPTCP in the Linux kernel after enabling MPTCP in the operating system or using the NS-3 simulator. Very few works implemented MPTCP on the CORE emulator.

A. Simulation

Chihani et al. [73] implemented MPTCP in the NS-3 simulator and introduced a new protocol that worked better in various network conditions. They compared different packet reordering systems and analyzed that their implementations will be necessary for further MPTCP performance analysis in terms of controlling congestion. Nadeem et al. [74] worked on introducing three path managers; default, ndiffports, and fullmesh to create an MPTCP patch for implementing MPTCP in the NS-3 development version. While the default patch has not made any new subflows, fullmesh made a mesh of

whole new subflows towards the feasible pairs of IP addresses; ndiffports introduced subflows in between the same IP pair with the help of distinct source and destination. It showed better results in terms of getting higher throughput and less flow completion time than prior works. Coudron et al. [75] proposed MPTCP implementations in NS-3 to handle network traffic. They also compared their algorithm with previous work implemented in NS-3 and Kernel. Table I lists some other MPTCP implementations in NS-2, NS-3 and CORE simulator with the aim of congestion control or schedule packets or perform both congestion control and packet scheduling for MPTCP.

B. Real Hardware (kernel)

Network simulators sometimes fail to depict the original network conditions, as the real-world network is highly dynamic; the breaking of links and the creation of new links are spontaneous. Therefore, the evaluation of MPTCP on real-world networks using Linux kernels shows a much bigger picture of its strengths and weaknesses. In the work [76], the authors have implemented MPTCP in a Linux kernel to study if each subflow has a different scheduler and then how the different subflows of an MPTCP may dispute bottleneck links with conventional single-path TCP. They tested LIA, OLIA, BALIA and wVegas on Linux kernel implementation of MPTCP and evaluated the throughput, latency, etc., on real-world networks. Zannettou et al. [24] used the kernel implementation of MPTCP to show their MPTCP-aware scheduling performs better than random hashing of packets to subflows which is generally used. They used the FatTree [77] and Jellyfish [78] topologies to conduct their experiments. FatTree is a highly structured topology used in data centers to obtain the highest throughput cost-effectively, while Jellyfish is the most commonly used randomly structured topology which can support more hosts than the FatTree, while keeping almost the same throughput. The commercial application for MPTCP support is available online [79].

VIII. CONCLUSION

This paper focuses on two crucial concepts in MPTCP - congestion control and scheduling. The study shows how the most recent works fulfill the previous work gaps and mitigate the above two MPTCP issues using different classical and ML-based approaches. Our study also presents the advantages and limitations of current works and encourages the researchers to continue further improvements in this domain. As in every communication sector MPTCP establishes a tremendous role, it is necessary to improve the performance of MPTCP, and our paper can be beneficial for the readers to have an extensive knowledge of MPTCP performance issues and can use it for proposing new algorithms.

REFERENCES

[1] L. G. Roberts, "The evolution of packet switching," *Proceedings of the IEEE*, vol. 66, no. 11, pp. 1307–1313, 1978.

[2] M. Hauben, "History of ARPANET," *Site de l'Instituto Superior de Engenharia do Porto*, vol. 17, pp. 1–20, 2007.

[3] J. Postel, "Rfc0768: User Datagram Protocol," 1980.

[4] J. Postel, "Transmission Control Protocol," tech. rep., Information Sciences Institute, University of Southern California, 1981.

[5] S. H. Baidya and R. Prakash, "Improving the performance of multipath TCP over heterogeneous paths using slow path adaptation," in *2014 IEEE International Conference on Communications (ICC)*, pp. 3222–3227, IEEE, 2014.

[6] G. Huston, "TCP in a wireless world," *IEEE Internet Computing*, vol. 5, no. 2, pp. 82–84, 2001.

[7] L. Chao, C. Wu, T. Yoshinaga, W. Bao, and Y. Ji, "A brief review of multipath TCP for vehicular networks," *Sensors*, vol. 21, no. 8, p. 2793, 2021.

[8] S. Hassayoun, J. Iyengar, and D. Ros, "Dynamic window coupling for multipath congestion control," in *2011 19th IEEE International Conference on Network Protocols*, pp. 341–352, IEEE, 2011.

[9] 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.

[10] A. Walid, Q. Peng, J. Hwang, and S. Low, "Balanced linked adaptation congestion control algorithm for MPTCP," *Internet Engineering Task Force, Internet-Draft draft-walid-mptcp-congestion-control-04*, 2016.

[11] R. K. Chaturvedi and S. Chand, "An adaptive and efficient packet scheduler for multipath TCP," *Iranian Journal of Science and Technology, Transactions of Electrical Engineering*, vol. 45, pp. 349–365, 2021.

[12] T. A. Le, C. S. Hong, M. A. Razzaque, S. Lee, and H. Jung, "ecMTCP: An energy-aware congestion control algorithm for multipath TCP," *IEEE Communications Letters*, vol. 16, no. 2, pp. 275–277, 2011.

[13] Y. Cao, M. Xu, and X. Fu, "Delay-based congestion control for multipath TCP," in *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pp. 1–10, IEEE, 2012.

[14] X. Ji, B. Han, R. Li, C. Xu, Y. Li, and J. Su, "ACCeSS: adaptive QoS-aware congestion control for multipath TCP," in *2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS)*, pp. 1–10, IEEE, 2022.

[15] T. Ha, A. Masood, W. Na, and S. Cho, "Intelligent multi-path TCP congestion control for video streaming in internet of deep space things communication," *ICT Express*, 2023.

[16] H. Wu, O. Alay, A. Brunstrom, G. Caso, and S. Ferlin, "Falcon: Fast and accurate multipath scheduling using offline and online learning," *arXiv preprint arXiv:2201.08969*, 2022.

[17] J. Han, K. Xue, J. Li, R. Zhuang, R. Li, R. Yu, G. Xue, and Q. Sun, "EdAR: An experience-driven multipath scheduler for seamless handoff in mobile networks," *IEEE Transactions on Wireless Communications*, 2023.

[18] S. J. Siddiqi, F. Naeem, S. Khan, K. S. Khan, and M. Tariq, "Towards AI-enabled traffic management in multipath TCP: A survey," *Computer Communications*, vol. 181, pp. 412–427, 2022.

[19] M. Y. Asiri, "A survey of multipath TCP scheduling schemes: Open challenges and potential enablers." https://www.techrxiv.org/, 2021.

[20] P. Tomar, G. Kumar, L. P. Verma, V. K. Sharma, D. Kanellopoulos, S. S. Rawat, and Y. Alotaibi, "CMT-SCTP and MPTCP multipath transport protocols: A comprehensive review," *Electronics*, vol. 11, no. 15, p. 2384, 2022.

[21] C. Xu, J. Zhao, and G.-M. Muntean, "Congestion control design for multipath transport protocols: A survey," *IEEE communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2948–2969, 2016.

[22] A. Jasin, R. Alsaqour, M. S. Abdelhaq, O. Alsukour, and R. Saeed, "Review on current transport layer protocols for TCP/IP model," *International Journal of Digital Content Technology and its Applications*, vol. 6, no. 14, pp. 495–503, 2012.

[23] W. Wei, K. Xue, J. Han, D. S. Wei, and P. Hong, "Shared bottleneck-based congestion control and packet scheduling for multipath TCP," *IEEE/ACM Transactions on Networking*, vol. 28, no. 2, pp. 653–666, 2020.

[24] S. Zannettou, M. Sirivianos, and F. Papadopoulos, "Exploiting path diversity in datacenters using MPTCP-aware SDN," in *2016 IEEE Symposium on Computers and Communication (ISCC)*, pp. 539–546, IEEE, 2016.

[25] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, no. 02, pp. 107–116, 1998.

[26] Z. Xu, J. Tang, C. Yin, Y. Wang, and G. Xue, "Experience-driven congestion control: When multi-path TCP meets deep reinforcement

learning," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 6, pp. 1325–1336, 2019.

[27] B. He, J. Wang, Q. Qi, H. Sun, J. Liao, C. Du, X. Yang, and Z. Han, "DeepCC: Multi-agent deep reinforcement learning congestion control for multi-path TCP based on self-attention," *IEEE Transactions on Network and Service Management*, vol. 18, no. 4, pp. 4770–4788, 2021.

[28] M. L. Puterman, "Markov decision processes," *Handbooks in Operations Research and Management Science*, vol. 2, pp. 331–434, 1990.

[29] G. Tesauro *et al.*, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[30] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI Open*, 2022.

[31] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[32] V. Pramanik and M. Maliha, "Analyzing sentiment towards a product using DistilBERT and LSTM," in *2022 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, pp. 811–816, IEEE, 2022.

[33] S. Deng, R. Netravali, A. Sivaraman, and H. Balakrishnan, "WiFi, LTE, or both? Measuring multi-homed wireless internet performance," in *Proceedings of the 2014 Conference on Internet Measurement Conference*, pp. 181–194, 2014.

[34] S. Singh, P. Mudgal, P. Chaudhary, and A. K. Tripathi, "Comparative analysis of packet loss in extended wired LAN environment," *International Journal of Computer Applications*, vol. 117, no. 2, 2015.

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

[36] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A scalable and flexible data center network," in *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, pp. 51–62, 2009.

[37] "Fat-Tree Design." https://clusterdesign.org/fat-trees/. [Accessed 19-Jul-2023].

[38] K. T. Hanna and P. Loshin, "NACK (NAK, negative acknowledgment, not acknowledged)," *TechTarget*, Aug 2021.

[39] F. Yang, Q. Wang, and P. D. Amer, "Out-of-order transmission for in-order arrival scheduling for multipath TCP," in *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, pp. 749–752, IEEE, 2014.

[40] Y.-C. Chen, Y.-s. Lim, R. J. Gibbens, E. M. Nahum, R. Khalili, and D. Towsley, "A measurement-based study of multipath TCP performance over wireless networks," in *Proceedings of the 2013 cConference on Internet Measurement Conference*, pp. 455–468, 2013.

[41] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[42] O. Ait-Hellal and E. Altman, "Analysis of TCP vegas and TCP reno," *Telecommunication Systems*, vol. 15, no. 3-4, pp. 381–404, 2000.

[43] J. Yang, J. Han, K. Xue, Y. Wang, J. Li, Y. Xing, H. Yue, and D. S. Wei, "TCCC: a throughput consistency congestion control algorithm for MPTCP in mixed transmission of long and short flows," *IEEE Transactions on Network and Service Management*, 2023.

[44] 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*, pp. 121–135, 2020.

[45] M. H. Rehmani and Y. Saleem, "Network simulator NS-2," in *Encyclopedia of Information Science and Technology, Third Edition*, pp. 6249–6258, IGI Global, 2015.

[46] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the NS-3 simulator," *SIGCOMM Demonstration*, vol. 14, no. 14, p. 527, 2008.

[47] X. Li *et al.*, "Using" random forest" for classification and regression.," *Chinese Journal of Applied Entomology*, vol. 50, no. 4, pp. 1190–1197, 2013.

[48] A. Singh, M. Xiang, A. Konsgen, C. Goerg, and Y. Zaki, "Enhancing fairness and congestion control in multipath TCP," in *6th joint IFIP Wireless and Mobile Networking Conference (WMNC)*, pp. 1–8, IEEE, 2013.

[49] S. Ferlin, Ö. Alay, T. Dreibholz, D. A. Hayes, and M. Welzl, "Revisiting congestion control for multipath TCP with shared bottleneck detection,"

[50] J. Ahrenholz, C. Danilov, T. R. Henderson, and J. H. Kim, "CORE: A real-time network emulator," in *MILCOM 2008-2008 IEEE Military Communications Conference*, pp. 1–7, IEEE, 2008.

[51] T. Zhang and S. Mao, "Machine learning for end-to-end congestion control," *IEEE Communications Magazine*, vol. 58, no. 6, pp. 52–57, 2020.

[52] R. Zhuang, J. Han, K. Xue, J. Li, D. S. Wei, R. Li, Q. Sun, and J. Lu, "Achieving flexible and lightweight multipath congestion control through online learning," *IEEE Transactions on Network and Service Management*, vol. 20, no. 1, pp. 46–59, 2022.

[53] A. Slivkins *et al.*, "Introduction to multi-armed bandits," *Foundations and Trends® in Machine Learning*, vol. 12, no. 1-2, pp. 1–286, 2019.

[54] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning*, pp. 387–395, Pmlr, 2014.

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

[56] J. Xu and B. Ai, "Deep reinforcement learning for handover-aware MPTCP congestion control in space-ground integrated network of railways," *IEEE Wireless Communications*, vol. 28, no. 6, pp. 200–207, 2021.

[57] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pp. 1871–1879, IEEE, 2018.

[58] S. R. Pokhrel, L. Pan, N. Kumar, R. Doss, and H. L. Vu, "Multipath TCP meets transfer learning: A novel edge-based learning for industrial IoT," *IEEE Internet of Things Journal*, vol. 8, no. 13, pp. 10299–10307, 2021.

[59] J. Hwang and J. Yoo, "Packet scheduling for multipath TCP," in *2015 Seventh International Conference on Ubiquitous and Future Networks*, pp. 177–179, IEEE, 2015.

[60] C. Paasch, S. Ferlin, O. Alay, and O. Bonaventure, "Experimental evaluation of multipath TCP schedulers," in *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing Workshop*, pp. 27–32, 2014.

[61] M. Scharf and S. Kiesel, "NXG03-5: Head-of-line blocking in TCP and SCTP: Analysis and measurements," in *IEEE Globecom 2006*, pp. 1–5, IEEE, 2006.

[62] P. Dong, J. Xie, W. Tang, N. Xiong, H. Zhong, and A. V. Vasilakos, "Performance evaluation of multipath TCP scheduling algorithms," *IEEE Access*, vol. 7, pp. 29818–29825, 2019.

[63] N. Kuhn, E. Lochin, A. Mifdaoui, G. Sarwar, O. Mehani, and R. Boreli, "DAPS: Intelligent delay-aware packet scheduling for multipath transport," in *2014 IEEE International Conference on Communications (ICC)*, pp. 1222–1227, IEEE, 2014.

[64] F. Yang, Q. Wang, and P. D. Amer, "Out-of-order transmission for in-order arrival scheduling for multipath TCP," in *2014 28th International Conference on Advanced Information Networking and Applications Workshops*, pp. 749–752, IEEE, 2014.

[65] S. Ferlin, Ö. Alay, O. Mehani, and R. Boreli, "BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks," in *2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pp. 431–439, IEEE, 2016.

[66] T.-A. Le and L. X. Bui, "Forward delay-based packet scheduling algorithm for multipath TCP," *Mobile Networks and Applications*, vol. 23, no. 1, pp. 4–12, 2018.

[67] F. Silva, M. Togou, and G.-M. Muntean, "An innovative machine learning approach to improve MPTCP performance," in *2020 International Conference on High Performance Computing and Simulation*, IEEE, 2020.

[68] X. Su, X. Yan, and C.-L. Tsai, "Linear regression," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 4, no. 3, pp. 275–294, 2012.

[69] T. M. Mitchell, "Artificial neural networks," *Machine Learning*, vol. 45, no. 81, p. 127, 1997.

[70] T. Viernickel, A. Froemmgen, A. Rizk, B. Koldehofe, and R. Steinmetz, "Multipath QUIC: A deployable multipath transport protocol," in *2018 IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, 2018.

[71] S. R. Pokhrel and A. Walid, "Learning to harness bandwidth with multipath congestion control and scheduling," *IEEE Transactions on Mobile Computing*, 2021.

[72] Q. Peng, A. Walid, and S. H. Low, "Multipath TCP algorithms: Theory and design," *ACM SIGMETRICS Performance Evaluation Review*, vol. 41, no. 1, pp. 305–316, 2013.

[73] B. Chihani and C. Denis, "A Multipath TCP model for ns-3 simulator," *arXiv preprint arXiv:1112.1932*, 2011.

[74] K. Nadeem and T. M. Jadoon, "An NS-3 MPTCP implementation," in *Quality, Reliability, Security and Robustness in Heterogeneous Systems: 14th EAI International Conference, Qshine 2018, Ho Chi Minh City, Vietnam, December 3–4, 2018, Proceedings 14*, pp. 48–60, Springer, 2019.

[75] M. Coudron and S. Secci, "An implementation of multipath TCP in NS3," *Computer Networks*, vol. 116, pp. 1–11, 2017.

[76] B. Y. L. Kimura and A. A. F. Loureiro, "MPTCP linux kernel congestion controls," *arXiv preprint arXiv:1812.03210*, 2018.

[77] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 63–74, 2008.

[78] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pp. 225–238, 2012.

[79] C. Paasch and S. Barre, "Multipath TCP in the linux kernel." available from http://www.multipath-tcp.org.