An Online Learning Assisted Packet Scheduler for MPTCP in Mobile Networks

Yitao Xing, *Graduate Student Member, IEEE*, Kaiping Xue, *Senior Member, IEEE*, Yuan Zhang, Jiangping Han, *Member, IEEE*, Jian Li, *Member, IEEE*, and David S. L. Wei, *Life Senior Member, IEEE, Member, ACM*

***Abstract*— Multipath TCP is designed to utilize multiple network paths to achieve improved throughput and robustness against network failure. These features are supposed to make MPTCP preferable to single-path TCP in mobile networks. However, it fails to achieve the expected performance in practice. A key challenge of using MPTCP in mobile networks is how to effectively spread packets over heterogeneous and unstable network paths to mobile devices with limited buffers. If packets are not sent in an effective way, MPTCP may only provide equal or even lower throughput than single-path TCP. Several packet scheduling algorithms have been designed to tackle this challenge. Unfortunately, they still cannot achieve the expected performance in dynamic scenarios such as mobile networks. In this paper, we propose an Online-Learning Assisted Packet Scheduler (OLAPS) to solve the packet scheduling problem by modeling it as a multi-armed bandit problem. Over time, OLAPS can adaptively learn from current network conditions to make the best scheduling policy to provide the highest possible throughput in a dynamic environment. Moreover, when the inbuilt reward monitor detects the mismatch between network conditions and the learned policy, OLAPS aborts the outdated policy and switches to a new one swiftly. We implement OLAPS as a Linux kernel module and evaluate it over a wide range of *ns-3*-simulated network conditions. The results show that OLAPS retains MPTCP's ability to provide higher throughput and also significantly improves the throughput performance of MPTCP when other in-kernel schedulers suffer a dramatic throughput decline.**

***Index Terms*— Multipath TCP, mobile networks, packet scheduling, online-learning.**

I. Introduction

NOWADAYS, mobile networks carry a significant fraction of the internet traffic, and providing satisfactory transmission service for users over mobile networks has become an urgent and challenging issue. According to Cisco [1], over 70 percent of the global population will have mobile connectivity by 2023, and the number of public Wi-Fi and Wi-Fi 6 hotspots will also rapidly increase. With mobile devices equipped with multiple network interfaces, users can now access the Internet anytime, anywhere with various wireless access technologies such as 4G, 5G, Wi-Fi, and Wi-Fi 6. However, the reliability of mobile networks has been largely overlooked. Frequent network failures [2], user mobility, network congestion, random packet loss, and other issues make the performances promised by the mobile networks unfulfilled.

To provide more stable and efficient transmission service over mobile networks, one feasible method is to concurrently utilize multiple network paths (e.g., Wi-Fi and LTE) between peers. In this way, a connection will not break down because certain network paths fail and can reach higher throughput by spreading data among multiple paths. Such a design is deemed a multipath transport protocol. There have been some transport-layer multipath protocols such as CMT-SCTP [3], multipath QUIC (MPQUIC) [4] and multipath TCP (MPTCP) [5]. Among them, MPTCP, as an extension of the most commonly used TCP, is now an IETF standard [5] and has been adopted by some major smartphone vendors such as Apple, Samsung, and Huawei [6], [7], [8]. With proper configuration, MPTCP can offer performance enhancements such as smoother network handover, resilience to network failures, and aggregated bandwidth of multiple network interfaces [9], [10], [11].

To achieve the enhancements, MPTCP should be configured properly according to network conditions and transmission requirements. The packet scheduler is the most crucial among the configurations. An MPTCP packet scheduler determines the way to distribute data over multiple network paths. Thus, different packet schedulers may lead to distinct transmission performances and should be carefully chosen. For example, managers may use the MinRTT scheduler (a.k.a. the "default" scheduler in the MPTCP Linux kernel [12]) to achieve higher throughput than single-path TCP, or they can use ReMP [13] (a.k.a. the "redundant" scheduler in the kernel) to pursue low transmission latency. Besides, several well-designed schedulers have been proposed to make fine-grained scheduling decisions. For example, BLEST [14] estimates and minimizes potential head-of-line (HoL) blocking in the receive window. ECF [15] estimates the arrival time of packets and sends packets on the path at which they will arrive earlier.

However, one specific hard-wired scheduler can hardly guarantee MPTCP's performance in the ever-changing network conditions in real-world mobile scenarios. That is because an

Manuscript received 26 February 2022; revised 27 August 2022 and 26 December 2022; accepted 13 February 2023; approved by IEEE/ACM Transactions on Networking Editor M. Zhang. Date of publication 22 February 2023; date of current version 17 October 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 61972371, in part by the Youth Innovation Promotion Association of Chinese Academy of Sciences (CAS) under Grant Y202093, and in part by the Fundamental Research Funds for the Central Universities. *(Corresponding author: Kaiping Xue.)*

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

Yuan Zhang is with the Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei, Anhui 230027, China.

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

Digital Object Identifier 10.1109/TNET.2023.3246168

optimal scheduling strategy highly depends on the characteristics of the paths being used [16]. Moreover, even in the same network, a scheduler may perform distinctly when the receivers have different buffer sizes. For instance, MinRTT works well in homogeneous networks, but in heterogeneous scenarios, it will cause many out-of-order (OFO) packets that have to be stored in the receivers' buffer. When the receivers have large buffers, OFO packets may bring some minor effects, such as long reorder delay and high buffer usage. But buffer-limited mobile devices (e.g., smartphones) may suffer dramatic throughput decline because they have to drop some packets when their receive buffers are full. Although schedulers like BLEST are supposed to solve this issue by design, their performances may be unpredictable with changing network conditions and random packet loss.

This paper aims to design an MPTCP scheduler that suits various network scenarios and provides consistently quality-guaranteed transmission service for mobile users. To this end, we propose an Online Learning Assisted Packet Scheduler (OLAPS) for MPTCP to adaptively generate scheduling policies. The proposed OLAPS solves the packet scheduling problem with a lightweight online-learning algorithm in multi-armed bandit (MAB) scenario named UCB1 [17]. In general, the UCB1-based OLAPS can learn a scheduling policy that maximizes the overall throughput overtime without any preliminary knowledge.

Simply adopting the UCB1 algorithm in OLAPS is quite inadequate in our work, and some important issues still need to be addressed. Firstly, we need to design a *reward function* that can help UCB1 chase high throughput in the context of multipath data transmission. Our designed reward function is based on the instantaneous throughput at both the subflow level and the connection level rather than depending on some inaccurate parameter measurements (e.g., congestion window, round-trip time, loss rate, etc.) in dynamic networks. Secondly, we introduce the *actions* of sending a certain number of redundant packets to eliminate the negative effects of delayed or lost packets on certain subflows. Thirdly, since UCB1 may lose its effectiveness in dynamic mobile networks, we further design a reward monitoring mechanism, which can precisely detect network changes and restart the learning process to generate a new policy.

We implement OLAPS as a module of MPTCP Linux kernel v0.95 [12] and evaluate it in our semi-physical simulation testbed composed of hosts with OLAPS and *ns-3* [18] networks. The results show that OLAPS can adapt to a wide range of network scenarios swiftly and significantly improve MPTCP's performance in various harsh network scenarios.

The main contributions of this paper are summarized as follows.

- We formulate the complex packet scheduling problem and solve it with a deployable online-learning scheme that adaptively generates better scheduling policies. Specifically, we design an intelligent scheduling agent with a reinforcement learning method in the multi-armed bandit (MAB) scenario. We further design a throughput-driven reward function and introduce different "redundant ratios" as the actions for the agent to pursue high instantaneous throughput.
- Then, we propose a reward monitoring mechanism to avoid the online-learning algorithm failure in dynamic mobile networks and speed up the learning process. This mechanism detects network changes by monitoring the deviation of rewards yielded by each action. When it detects a network change, it restarts the learning algorithm to explore a new optimal policy.
- Based on different online-learning methods, we implement different OLAPS variants as modules for an MPTCP-enabled Linux kernel. Rich experiments are conducted to evaluate OLAPS variants and existing heuristic schedulers in a semi-physical simulation testbed. This testbed combines hosts with actual Linux kernel and controllable *ns-3*-simulated networks to get realistic and statistically significant results. In the experiments, OLAPS variants show their adaptability and significant improvement of MPTCP's performances to different network scenarios.

The rest of this paper is organized as follows. In Section II, we briefly introduce MPTCP's packet scheduling problem and the adversarial bandit problem as its solution. In Section III, we review some state-of-art scheduling algorithms. Then we formulate the packet scheduling problem and present a solution in Section IV. The design and implementation of OLAPS are given in Section V. After that, we discuss some key designs of OLAPS in Section VI. In Section VII, we present our evaluation of OLAPS and its comparison with other in-kernel schedulers. We discuss the limitations of our work as well as our future work in Section VIII. Finally, a conclusion is drawn in Section IX.

II. Background and Motivation

In this section, we first present the overview of MPTCP's packet scheduling problem. Then, we introduce an online learning approach as a solution. Finally, we discuss the motivation for designing OLAPS.

A. MPTCP and the Packet Scheduling Problem

Multipath TCP (MPTCP) enables a transport-layer connection to operate simultaneously across multiple network paths (or subflows). This is a desirable feature for multi-homed devices in mobile networks because by using multiple subflows concurrently, MPTCP is expected to achieve higher throughput [19], [20] and enhanced resilience to network failures [10], [11]. Besides, MPTCP, as a standardized extension of regular TCP [5], is transparent to both upper and lower layers, which makes it deployable in real networks. However, inappropriate packet schedulers may prevent MPTCP from performing as well as expected [21], [22]. An MPTCP packet scheduler decides on which path a packet should be sent, significantly impacting the final transmission performance [23]. As a result, the proper design of a packet scheduler and which scheduler should be used in different scenarios are urgent issues to be solved.

Several heuristic schedulers have been designed to achieve different goals, such as high throughput, low latency [13], HoL blocking elimination [14], [15], [24], [25], and so on. However, none of these hard-wired schedulers can provide consistent and expected performance under various scenarios

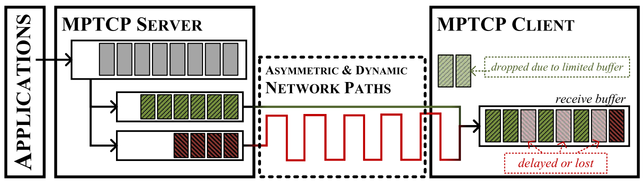


Fig. 1. The packet scheduling problem of MPTCP.

in real mobile networks [26]. A typical example is Min-RTT, which is designed to achieve higher throughput and lower transmission latency by filling the congestion window (CWND) of the subflow with the lowest round-trip time (RTT) before other subflows in ascending RTT order. It works well with symmetric network paths but has been proven suboptimal otherwise. This is because when paths are asymmetric (e.g., with much different RTT), packets are likely to arrive out-of-order [27], and the receiver needs to queue them in the receive buffer as shown in Fig. 1. This issue is called head-of-line (HoL) blocking, meaning that the early-arrived packets have to wait for stalled packets with lower sequence numbers from a subflow with higher RTT. The HoL blocking issue causes prolonged reorder delay and even packet drop if packets exceed the buffer size limit. In some cases, single-path TCP may even outperform MPTCP due to such an issue, which makes using MPTCP in the real network a risk-taking option. Though schedulers such as BLEST [14] and ECF [15] try to solve this issue, their performances in dynamic network scenarios are still questionable [28].

Given this unsolved issue, MPTCP is usually used in a conservative *backup mode*. For example, Apple's devices establish a primary subflow over Wi-Fi and a backup subflow over cellular data. They only use the backup subflow when the primary subflow becomes unavailable [7]. Such a stopgap measure can not give full play to the advantages of MPTCP. In conclusion, the packet scheduling problem has become an obstacle to adopting MPTCP in mobile networks.

B. The Multi-Armed Bandit Problem

To tackle the challenges mentioned above, designing an intelligent scheduler that can adaptively generate suitable scheduling policies is a desired solution. In this article, we formulate the packet scheduling problem as a classic MAB problem [17], [29], [30], [31]. This well-studied problem provides a simple model for the exploration versus exploitation dilemma, which can be described as the search for a balance between exploring the environment to find profitable actions while taking the empirically best action as often as possible. As a result, it is widely used to solve many sequential decision-making tasks.

Generally, in a K-armed bandit problem, a gambler must choose an arm of K-slot machines to play in a sequence of trials. Plays of machine i in trials $(t_1, t_2, \dots)$ yield rewards $(x_i(t_1), x_i(t_2), \dots)$, which are independent and identically distributed according to an unknown law with unknown expectation. A policy is an algorithm that chooses the next machine to play based on the sequence of past plays and obtained rewards. The gambler's goal is to maximize the summarized reward with a policy that can find a good balance between exploiting the arm with the highest reward currently and exploring the other arms to find a possibly better one.

Given the description, the packet scheduling problem can perfectly suit the framework of the MAB problem. The scheduler can be regarded as the gambler that chooses one of the K scheduling strategies for periods repeatedly during the lifetime of an MPTCP connection. And at the end of each time period, the scheduler calculates the reward of the period with a reward function. A well-designed reward function reflects the performance of the chosen strategy, so the goal of achieving better transmission performance is equivalent to that of maximizing the summarized reward. To maximize the summarized reward, the scheduler also faces the exploration versus exploitation dilemma, just like the gambler in the MAB problem. Therefore, leveraging the solutions for the MAB problem is a good idea to solve the packet scheduling task.

C. The Goals of OLAPS

In this subsection, we present two design goals of OLAPS.

The first goal of OLAPS is to **generate adaptive policies in scheduling** to achieve the highest possible instantaneous throughput under different network scenarios. Hard-wired scheduling algorithms can hardly perform consistently well in real mobile networks due to dynamic network conditions and buffer-limited mobile devices. Therefore, the ability to generate adaptive policies is a must. On the other hand, high instantaneous throughput guarantees that applications perform well during a complete MPTCP connection. For example, high overall throughput may be suitable for tasks such as file download because a declined throughput over a short period of time may only slightly affect the overall user experience. However, applications like video streaming may suffer from low video resolution or even pauses due to insufficient instantaneous throughput at some point, even though the average throughput is beyond the requirements. To achieve this goal, we formulate the packet scheduling problem as a MAB problem and design an intelligent scheduler based on online-learning methods. Furthermore, we carefully construct the reward function and the actions in such an MAB problem in the context of a transmission process.

However, the online-learning algorithms may suffer a slow convergence when network conditions change abruptly, which may lead to suboptimal scheduling policies during the long convergence time. As a result, the second goal of OLAPS is to **respond swiftly to abrupt network changes**. Specifically, in an MAB solution, the gambler exploits the highly rewarding arms at the moment. As a result, when the reward distribution behind the slot machine changes, the preferred arms cannot provide high rewards as before. A gambler will randomly choose other arms with a certain probability to explore potential high rewards to avoid such a dilemma. However, he may take a long period to replace the outdated policy with a better one. To solve this problem, OLAPS deploys a network monitor and detects abrupt reward variations, and starts a new epoch of the online learning process when network changes are detected.

We present the details of the design and implementation of OLAPS in Section V, and our evaluation in Section VII shows that achieving these two goals makes OLAPS a competent MPTCP scheduler in mobile networks.

III. Related Work

MPTCP breaks through the bandwidth limit of a single network interface and can provide better resilience to network failures by transferring data to available network paths. However, it also brings challenges that single-path protocols have never met [9], [10], [11]. One of the known issues is that concurrent use of multiple heterogeneous network paths may lead to declined performance. Specifically, packets sent through paths with different conditions may arrive out-of-order, leading to HoL blocking, long reorder delay, and receive buffer occupation. In some cases (e.g., receivers are mobile devices with limited buffer size), MPTCP may even underperform single-path TCP.

To solve this issue, researchers have proposed some packet scheduling algorithms for multipath transport protocols. In the MPTCP Linux kernel [12], there have been some simple schedulers such as round-robin (RR), MinRTT, and ReMP [13]. RR may be the simplest scheduler that evenly sends a packet to all available subflows in turn. It works well if subflows are symmetric but will encounter severe HoL blocking issues otherwise. MinRTT mitigates this issue by preferring the subflow with the lowest RTT. Still, the same issue may happen when it starts to use some slower subflows.

Some other schedulers try to eliminate the HoL blocking issue by making packets traveling through different subflows arrive in order. A common method is to estimate the arrival time of each packet with measurements such as delays and capacities of subflows. For example, DAPS (Delay-Aware Packet Scheduler [32]), a CMT-SCTP extension, makes scheduling decisions according to the one-way-delay and capacities of paths, but it is shown that it generates spurious retransmissions. BLEST (Blocking Estimation [25]) is designed to reduce buffer blocking by skipping a certain currently available path and waiting for a more advantageous path that can offer a lower risk of blocking. Besides, OTIAS (Out-of-Order Transmission for In-Order Arrival Scheduler [24]), ECF (Earliest Completion First [15]), DPSAF (Dynamic Packet Scheduling and Adjusting with Feedback) [33], and STTF (Shortest Transfer Time First [25]) try to estimate the arrival time more precisely by taking queuing delay in send buffer into consideration. Wei et al. [34] used another approach for precise arrival time estimations by combining the packet scheduling strategy with a certain congestion control algorithm. Despite these efforts, they can hardly provide consistently good performances in some network scenarios [26], [28] because it can be hard to estimate a packet's arrival time precisely in a dynamic network, and unpredictable random packet loss also makes such estimations unreliable.

There are other schedulers who pursue performance metrics other than throughput. RAVEN [35] is an in-kernel MPTCP scheduler that mitigates tail latency and network unpredictability by using redundant transmission. A recent MPQUIC packet scheduler proposed by Zheng et al. [36] takes the priority of contents into consideration. By leveraging the stream multiplexing feature of MPQUIC, it uses a QoE-driven scheduling design to speed up the loading of certain high-priority contents.

Instead of designing a packet scheduling algorithm, Pokhrel and Mandjes [37] designed a delay-adaptive congestion control algorithm that controls the reordering delay at the receiver by taking into account the loss and delay characteristics of the network paths. Further, they [38] adopted the rent-seeking framework to control the sending rate of each subflow in such a way that packets are more likely to arrive in order.

Though hard-wired heuristic schemes can improve the performance of transmission in some specific cases, they do not have the adaptability to perform consistently well in a wide range of network scenarios. Given the limitation of heuristic methods, many machine-learning schemes are proposed, empowering TCP as well as MPTCP with intelligence to adapt to various scenarios. For example, Pokhrel et al. [39] developed a novel multipath communication framework for Industry 4.0 using an experience-driven Deep Q-Network (DQN), to achieve human-level intelligence in networking automation and orchestration. And to improve the performance of the Internet of Vehicles, a federated learning framework [40] was proposed to enhance TCP performance in networks with unstable RTT and packet loss. Abbasloo et al. [41] used deep reinforcement learning techniques to steer throughput-oriented TCP algorithms and boost the performance of various old and new TCP congestion control algorithms. Huang et al. [42] proposed a distributed Deep Reinforcement Learning (DRL)-based congestion control algorithm to realize objective-oriented resource pooling in MPTCP. Chung et al. [43] designed a machine learning scheme to manage the usage of multiple network paths.

As one of the key components of transmission, machine learning empowering multipath packet scheduling algorithms can also largely improve MPTCP performance. Unlike the hard-wired schedulers above, learning-based schedulers are proposed to generate adaptive policies for various network scenarios. ReLeS [26] is the first learning-based MPTCP packet scheduler. It formulates the multipath packet scheduling problem as a reinforcement learning task and solves the task with asynchronous deep reinforcement learning techniques [44]. Specifically, ReLeS applies DQN to teach the scheduler with data collected from each finished MPTCP connection. Though ReLeS can achieve great performance with a well-trained neural network, the need for training data and the overhead of training a DQN make its wide deployment a challenge. Wu et al. [28] presented a learning-based scheduling algorithm named Peekaboo for MPQUIC, which tackles the scheduling problem using an online learning method based on contextual MAB theory [45]. Peekaboo decides to skip some available paths and wait for a more advantageous one using LinUCB algorithm [46]. Though ReLeS [26] and Peekaboo [28] are the most relevant work for OLAPS, they are quite different and are not mutually comparable. Specifically, ReLeS needs a neural network that is well-trained offline, while OLAPS is an online-learning method that needs no training data. And Peekaboo is an MPQUIC scheduler that chooses the best subflow to send a packet, while OLAPS is an MPTCP scheduler that decides to send which kind of packets (redundant or not). Therefore, we did not choose these two schedulers for comparison since it does not help evaluate OLAPS.

In this paper, our design of OLAPS is based on MPTCP because this IETF standard protocol has been adopted by many vendors, and it is now an integral part of 5G mobile networks as a standard feature of 3GPP Release 16 [47]. Also, since

MPTCP is a Linux kernel module, OLAPS can run on any MPTCP-enabled Linux kernel for testing.

IV. Problem Formulation and a Solution

A. Overview

We describe the multipath packet scheduling problem in the multi-armed bandit problem framework [29] in this section.

In this paper, a multipath packet scheduler is regarded as a gambler, and different scheduling policies are the arms of slot machines to pull. The scheduler gets packets from an application and chooses one of the K scheduling strategies. After choosing a certain strategy (just like pulling a certain arm), the scheduler sends packets to multiple network paths for a *scheduling interval* following the chosen policy and gets acknowledgments (ACKs) from the receiver afterward. Then the scheduler can calculate the throughput in this interval and takes it as the reward of the chosen strategy. The goal of the scheduler is to reach the maximum throughput possible.

B. Formulation of Multipath Packet Scheduling Problem

We divide time into a series of intervals, called *scheduling intervals* (SIs), and an *epoch* contains several SIs. There may be several epochs in the duration of a complete MPTCP connection. In each SI, the scheduler takes one of the K possible actions stochastically, where each action is denoted by an integer $1 \leq i \leq K$. Only after an action is taken, the scheduler gets an assignment of *rewards*, which is an infinite sequence $\boldsymbol{x}(1), \boldsymbol{x}(2), \ldots$ of vector $\boldsymbol{x}(t) = (x_1(t), \ldots, x_K(t))$ where $x_i(t) \in [0,1]$ denotes the reward obtained if the scheduler takes action i at time step (or "trial") t. We further assume that the scheduler only knows the rewards $x_{i_1}(1), \ldots, x_{i_t}(t)$ of the previously chosen actions $i_i, \ldots, i_t$. The rewards of an action i are independent and identically distributed according to an unknown law with unknown expectation μ_i.

Denote A as ab algorithm that t chooses the next machine to play based on the sequence of past plays and obtained rewards. And let $T_I(n)$ be the number of times action i has been taken by A. Then the *regret* of A after n trials is defined by

$$\mu^* n - \mu_j \sum_{j=1}^{K} \mathbb{E}\left[T_j(n)\right], \tag{1}$$

where $\mu^* \overset{def}{=} \max_{1 \leq i \leq K} \mu_i$ and $\mathbb{E}[\cdot]$ is the expected value.

C. An Online Learning Solution

Given the problem we describe above, we adopt the **UCB1** algorithm proposed by Auer et al. [17] as the base of our scheduler. In this section, we introduce the UCB1 algorithm.

As shown in Algorithm 1, on each trail, UCB1 takes an action i that maximizes the sum of two terms. The first one is simply the average of action i so far. The second term is related to the size (according to Chernoff-Hoeffding bounds) of the one-sided confidence interval for the average reward and the true expected reward of the action falls within it with overwhelming probability. According to Auer et al. [17], for all $K > 1$, if UCB1 is run on K machines having arbitrary reward distributions $P_1, \ldots, P_K$ with support in $[0,1]$, then its expected regret after any number n of trials is at most

$$\left[8 \sum_{i:\mu_i<\mu^*} \left(\frac{\ln n}{\mu^* - \mu_i}\right)\right] + \left(1 + \frac{\pi^2}{3}\right)\left[\sum_{j=1}^{K} (\mu^* - \mu_j)\right], \tag{2}$$

where μ_i is the expected values of P_I and μ^* is any maximal element in the set $\mu_1, \ldots, \mu_K$.

In the next section, explain how UCB1 is actually used in packet scheduling process, and describe our design of the reward function and actions in the MAB problem to UCB1 work as a multipath packet scheduler.

Algorithm 1 UCB1

```
Parameters: $x_i \in [0,1]$, current reward of action $i$
            $\bar{x}_i \in [0,1]$, average reward of action $i$
            $n_i$, the number of times of action $j$
            $n$, the overall number of trials
1 Function Initialization():
2     Take each action once and set the parameters.
3 end
4 Function Main Loop():
5     Take action $i$ that maximizes $\bar{x}_i + \sqrt{\frac{2 \ln n}{n_i}}$
6     Get reward $x_i$.
7     Update the parameters.
8 end
```

V. Online Learning Assisted Packet Scheduler: Design and Implementation

A. Overview

In this section, we present our design and implementation of the Online-Learning-Assisted Packet Scheduler (OLAPS) for MPTCP. As shown in Fig. 2, OLAPS includes a UCB1-based *online learning agent*, a *reward monitor*, and a *packet scheduler* to achieve the goals to be discussed in Section II-C. The UCB1-based online learning agent (or UCB1 agent for short), as its name implies, runs the UCB1 algorithm to generate a scheduling policy, which is about choosing a "redundant ratio" for the current scheduling interval (SI). Then the agent collects helpful information from the acknowledgments (ACKs) to optimize its policy. Then the *packet scheduler* makes the final scheduling decisions: It finds a subflow with the lowest RTT and sends a duplicated or non-duplicated packet to it according to the chosen redundant ratio. Meanwhile, the reward monitor detects abrupt network changes and intervenes in the learning process of the agent for faster convergence.

At the beginning of each SI t, the UCB1 agent choose a "redundant ratio" p_i among K given options (i.e., $p_i, \ldots, p_K$). The redundant ratio determines the amount of duplicated packets sent on different subflows. Generally speaking, a smaller redundant ratio means sending fewer duplicated packets, which is a more aggressive policy that chases high throughput by aggregating bandwidth from multiple subflows. On the contrary, a larger redundant ratio means more packets are sent redundantly on all subflows, which is a more conservative

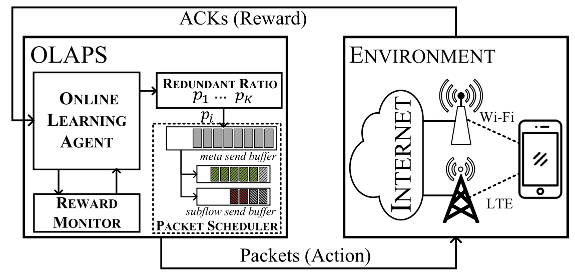


Fig. 2. The framework of OLAPS.

policy to avoid throughput degradation caused by delayed or lost packets. And at the end of this SI, the UCB1 agent receives ACKs and calculates a reward for the chosen redundant ratio as

$$x_i(t) = \frac{meta\_thp(t)}{\sum_{s=1}^{N} sub\_thp_s(t)}, \tag{3}$$

where $meta\_thp(t)$ denotes the throughput of the "meta socket" (i.e., the overall throughput of the MPTCP connection) in current SI t, and likewise, $sub\_thp_s(t)$ denotes the throughput of subflow s of all N subflows. A higher reward indicates the chosen action utilizes multiple subflows efficiently. Then the UCB1 agent runs Algorithm 1 and starts the next SI $t+1$. We may ignore the variable t when it is clear from the context for easier reading.

The UCB1 agent does not directly select a subflow to send a certain packet. Instead, it guides the *packet scheduler* to make final scheduling decisions by offering a redundant ratio. The packet scheduler prefers the subflow with the lowest RTT, which is similar to the MinRTT algorithm. However, different from the original MinRTT, the scheduler in OLAPS sends a certain proportion of redundant packets according to the redundant ratio provided by the UCB1 agent. Note that the scheduler may run other algorithms for subflow selection, but it should always send a certain amount of duplicated packets following the agent's guidance. In this paper, we choose MinRTT because it is simple yet efficient, according to previous research.

Though the UCB1 agent can generate adaptive scheduling policies, it may suffer from slow convergence because of noisy rewards and changing networks. Therefore, we design a reward monitoring mechanism in the OLAPS framework. Specifically, we deploy a reward monitor that filters the noisy reward samples and maintains the deviation between the reward samples. When current samples deviate a lot, the reward monitor determines that current network conditions have changed abruptly. Then it refreshes the learned model of the agent to learn a new policy instead of sticking to the outdated one.

For the rest of this section, we carefully describe the design and implementation of each component in OLAPS.

B. The UCB1-Based Online Learning Agent

This agent adopts the UCB1 algorithm to solve the MAB problem, and we need to properly design the reward function and the actions of the algorithm.

Reward: We expect the reward function to directly reflect the performance of the previously chosen action, and the throughput can be a good indicator. Since modeling MPTCP's throughput with measurements such as RTT, packet loss rate, and CWND is complicated and unreliable in dynamic mobile networks, we decide to record the actual throughput of the MPTCP connection ($meta\_thp$ for short) in each SI and use it as a component of the reward.

However, using $meta\_thp$ straightforwardly makes it hard to tell if the reward is changing because of taking different actions or experiencing different network conditions (e.g., link capacity decline). As a result, we divide $meta\_thp$ by the sum of the throughput of each subflow, and thus the reward function is in the form of Eq. 3. In this way, the agent can get the reward reflecting how the chosen action behaves in utilizing multiple subflows to achieve better performance. A higher reward indicates that the chosen action can effectively aggregate the capacities of multiple subflows and finally achieve higher throughput. When $meta\_thp$ decreases because of network congestion or failure that happens on certain subflows, an optimal action can still get a high reward because the denominator also decreases.

However, the sub_thp_s in Eq. 3 is still not well-defined. We notice that when inappropriate actions are taken, the throughput of subflows in the SI will sometimes decline due to dropped packets in receive buffer, which will unreasonably increase the rewards of such actions. To solve this issue, instead of using $sub\_thp_s(t)$, we calculate the reward as

$$x_i(t) = \frac{meta\_thp(t)}{\sum_{s=1}^{N} \max_{\tau \in [t-\mathcal{T},t]}(sub\_thp_s(\tau))}, \tag{4}$$

where $\max_{\tau \in [t-\mathcal{T},t]}(sub\_thp_s(\tau))$ denotes the maximum throughput of subflow s within a time window T.

Action: When the UCB1 agent is expected to take action, it chooses one of the K redundant ratios, i.e., $p_i = \frac{i-1}{K-1}$, where integer $i = 1, \ldots, K$, and integer $K \geq 2$. Then, a chosen redundant ratio p_i is applied in the current SI, which means p_i of all packets sent through a subflow should also be sent on other subflows.

The ability to send redundant packets is quite useful for MPTCP schedulers for two reasons. First, sending redundant packets avoids the impact of lost or delayed packets on subflows with poor conditions since their duplicates can arrive at the receiver safely through other subflows. Second, by sending packets on all available subflows, schedulers can get enough feedback to keep their knowledge of the subflows up-to-date. More discussions on the actions of OLAPS are given in Section VI.

C. The Reward Monitor

The UCB1 agent itself is not very efficient when dealing with dynamic network environments or networks with random loss. According to Auer et al. [17], the UCB1 algorithm assumes that the K machines have arbitrary reward distributions. However, in mobile networks, the reward distributions may easily change due to the dynamic nature of the networks. To understand why UCB1 is not suitable for mobile networks, simply consider the average reward of action i ($\bar{x}_i$ in Algorithm 1). When the distribution of action i changes due to network fluctuations, the $\bar{x}_i$ can hardly represent the expected reward. Then, the UCB1-based agent will consequently generate suboptimal policies.

Algorithm 2 Reward Monitor

```
Parameters : $\alpha \in (0,1), mul, thr, n$
Initialization: set $srwd = rwd$
                and $dev = 0.5 \times rwd$
Function CheckReward($rwd$, $i$):
    # For each obtained reward of action $i$
    if within the first $n$ trails then
        $dev = (1-\alpha) \times dev + \alpha \times abs(rwd - srwd)$
        return
    end
    if $rwd > arwd + mul \times dev$ then
        if $cnt < 0$ then
            $cnt = 0$
        end
        $cnt = cnt + 1$
    end
    if $rwd < arwd - mul \times dev$ then
        if $cnt > 0$ then
            $cnt = 0$
        end
        $cnt = cnt - 1$
    end
    $dev = (1-\alpha) \times dev + \alpha \times abs(rwd - srwd)$
    if $abs(cnt) > thr$ then
        # Restart the UCB1 algorithm
        Restart UCB1()
        # Initialize the monitor again
        Do Initialization()
    end
end
```

To solve this issue, we design a reward monitor to intervene in the UCB1 algorithm externally. The monitor follows Algorithm 2 and is triggered by every reward for each action. The reward monitor maintains a smoothed absolute deviation (dev) between the current reward sample rwd and the average reward $arwd$. When there are thr number of reward samples that deviate a lot from the average reward, the monitor resets the UCB1 agent by starting a new epoch of the UCB1 algorithm, which means setting $t = 1$ and redoing the initialization. Then, to learn the variation in the reward samples following a new distribution, the monitor initials the dev based on the first γ reward samples without triggering the reset.

D. The Overall OLAPS Implementation

Finally, we put all the components together and describe how they assist the packet scheduler in making final scheduling decisions. OLAPS is described in Algorithm 3. When two hosts initiate an MPTCP connection, OLAPS also starts to decide which subflow each packet will be sent on by following the guidelines of the UCB1 agent. OLAPS divides the lifetime of the MPTCP connection into many time periods, named scheduling intervals (SIs). At the beginning of each SI, OLAPS chooses a redundant ratio using the UCB1 algorithm, and then in this SI, OLAPS schedules packets according to the chosen policy and collects useful information from the packets and acknowledgments as the hosts communicate with each other. Finally, at the end of this period, OLAPS calculates the reward of the chosen action and updates the parameters of the UCB1 algorithm. Then it starts a new time period and repeats the above steps until the connection is closed.

Algorithm 3 OLAPS

```
Function SubflowSelection():
    # Adopt MinRTT
    find an available $subflow$ with the lowest RTT
    return $subflow$
end
Function PacketSelection($p_i$):
    if $red\_quota == 0$ and $new\_quota == 0$ then
        $red\_quota = p_i \times CWND$
        $new\_quota = CWND - red\_quota$
    end
    if $red\_quota > 0$ then
        Send a redundant packet.
        $red\_quota = red\_quota - 1$
    else
        Send a normal packet.
        $new\_quota = new\_quota - 1$
    end
    return the packet
end
Function GetReward():
    Get the throughput of at connection level ($meta\_thp(t)$).
    Get the throughput of each subflow $s$ ($sub\_thp_s$)
    Get a reward ($rwd$). # with Eq. 4
    CheckReward($rwd$) # Algorithm 2
    return $rwd$
end
Function AtSndingPkt():
    if current SI is ended or for initialization then
        Start a new SI.
        Take action $i$ following UCB1 in Algorithm 1.
        Redundant ratio $p_i = \frac{i-1}{K-1}$ for the SI.
    end
    subflow = SubflowSelection()
    packet = PacketSelection($p_i$)
    Send the packet to the subflow.
    Collect information to calculate the rewards.
end
Function AtRcvingPkt():
    Collect information to calculate the rewards.
    End the SI if time is up.
    if current SI is ended then
        $rwd$ = GetReward()
        Update the UCB1 parameters following Algorithm 1.
    end
end
```

TABLE I

PARAMETERS IN OLAPS IMPLEMENTATION AND THEIR TYPICAL VALUES

Symbol	Note	Typical Value
SI	duration of a scheduling interval	$3 \times$ the maximal RTT
K	the number of actions (Algorithm 1)	3
α	to smooth deviation (Algorithm 2)	0.125
mul	to check reward (Algorithm 2)	3
thr	the threshold of the reward monitor (Algorithm 2)	4
n	the number of trails without triggering reset (Algorithm 2)	$15 \times K$

To be specific, at any moment the MPTCP connection wants to send a packet, OLAPS runs `AtSndingPkt()`. OLAPS starts an SI by choosing a redundant ratio with the UCB1 agent. During the period, OLAPS first finds the subflow with the shortest round-trip time and an open congestion window. Then it decides whether the next packet sent on the subflow should be duplicated or not. To this end, OLAPS checks its congestion window size and the redundant ratio. A redundant ratio p_i indicates that to fill the $cwnd_s$-sized congestion window of each subflow s, OLAPS should firstly fill in $p_i \times cwnd_s$ number of duplicated packets that have been scheduled to other subflows, then fill the rest of the window with non-duplicated packets. OLAPS collects information such as the sequence number of packets and current time stamps from all subflows as well as the MPTCP connection. Therefore, OLAPS knows the exact time when an SI starts and how many bytes are sent during this SI.

Similarly, when the connection receives a packet, OLAPS runs `AtRcvingPkt()`. It records the acknowledged sequence number and updates the time stamp. When the lifetime of the current SI is up (the lifetime is three times the maximal RTT of all subflows), OLAPS calculates the reward for the chosen action and advances the UCB1 algorithm to the next trail. OLAPS repeatedly runs until the connection is closed. Specifically, to calculate the reward, the agent has to calculate the throughput of the MPTCP connection and each subflow (lines 21, 22 in Algorithm 3). In `AtSndingPkt()`, OLAPS can record the sequence numbers of the first and last transmitted packets and the timestamp at their sending and acknowledgment within an SI (line 39). By dividing the offset of the sequence numbers (i.e., the number of transmitted bytes) by the offset of the first sending timestamp and the last acknowledgment timestamp (i.e., the actual transmission time), we get the throughput. Since the packet sequence spaces at the MPTCP connection and subflow levels are separate, OLAPS can get the throughput at both levels to calculate the reward following Eq. 4.

Each MPTCP connection runs an individual OLAPS, and no modification is needed on the client side. Furthermore, since it is an online learning solution, no data is needed for training in advance. Leveraging the pluggable architecture for the packet scheduler [48] in the MPTCP Linux kernel [12], OLAPS is implemented as a kernel module with over 1000 lines of C code. The parameters and their typical values in the OLAPS implementation are summarized in Table I.

Based on the design of OLAPS, it may sometimes perform abnormally when the MPTCP connections use window-based congestion control algorithms that create bursts of packets and ACKs. OLAPS may collect inaccurate information due to these bursts and fail to learn an optimal scheduling policy. As a result, we choose the widely-studied BBR [49] as the congestion control algorithm in this paper. BBR paces every packet to match the data rate of the network bottleneck, and thus it works perfectly with OLAPS. Also, since one of its typical usage scenarios is with public Wi-Fi [49], BBR can be a promising algorithm in mobile networks to work with other existing schedulers.

VI. DISCUSSION

In this section, we discuss the key designs in OLAPS.

A. The Throughput-Driven Reward Function

In Section V-B, we present a "throughput-driven" reward function for the online-learning agent. The reason to use instantaneous throughput (i.e., the throughput in a short period or SI) as the component of the reward function is that it properly reflects the performance of the chosen action. Compared with the reward functions that are related to an extended period (e.g., discounted throughput [28] or average throughput), our design mainly focuses on the short-term performance of the scheduler in order to achieve swift scheduling policy adjustment in changing environments. As a result, such a design is more suitable for mobile network scenarios.

Another type of reward function tries to model the transmission performance with measurements such as RTT, packet loss rate, receive buffer size, congestion window, and so on. Such designs may cut down the reward when RTT and loss rate is high in order to avoid using network paths with bad conditions, which is a feasible approach according to previous work [26], [50]. However, these designs are unsuitable for OLAPS since OLAPS does not avoid using "bad" network paths. Instead, OLAPS sends redundant packets to them. Besides, it is hard to determine a precise relationship between the performance of an action and the measurements because the mobile network is a sophisticated and dynamic system. Worse still, the measurements can be inaccurate in this kind of dynamic environment and are unstable in a short period, such as in an SI.

Finally, we choose a reward function in the form of Eq. 4 for OLAPS, which can yield the rewards that reflect how the currently chosen action affects the instantaneous throughput of the MPTCP connection without using inaccurate measurements. In this way, OLAPS can generate optimal scheduling policies adaptively based on the rewards, which are highly affected by some specific factors, such as receive buffer size and network conditions. For example, when dealing with highly asymmetric network paths, the function may yield low rewards if the receiver has a limited buffer size that can be easily overflowed by out-of-order packets, leading to throughput reduction. Meanwhile, in the same network scenario, the function can also yield high rewards when the receive buffer is large enough to store the out-of-order packets.

B. The Actions

The actions of OLAPS are different in *redundant ratios*, which means sending a certain fraction of redundant packets on all other subflows. By deciding to take lower or higher redundant ratios, OLAPS basically chooses to act more like MinRTT or ReMP. Choosing a scheduling policy that performs the best under current network conditions can be a great improvement for MPTCP's overall transmission performance.

Intuitively, sending redundant packets will waste precious network resources and will finally cause low throughput. Thus choosing a lower or even zero redundant ratio (i.e., acting more like MinRTT) should be the best choice. That is true when the network paths are symmetric, or the receiver has a large enough buffer. However, when the subflows are asymmetric, e.g., with different RTTs, loss rates, and bandwidths, packets from different subflows may arrive out-of-order and occupy the limited receive buffer, which may lead to HoL blocking and finally cause buffer overflow for buffer-limited devices. These issues not only affect a single subflow that suffers harsh network conditions. What is worse is that the dropped or delayed packets from a certain subflow may adversely affect other well-behaved subflows, making MPTCP underperform single-path TCP in some cases.

Sending a certain amount of redundant packets, i.e., acting more like ReMP, can mitigate such an issue. That is because delaying or losing redundant packets will not affect the overall transmission performance of the MPTCP connection if they can be recovered with their copies from other subflows. Especially when all packets are sent redundantly to all subflows, OLAPS behaves just like ReMP. In this case, delayed or lost packets on the subflows can be replaced with their redundant copies from other subflows so that the subflows with high delay and packet loss will not affect the transmission of other subflows. As a result, the throughput of MPTCP is determined by the highest throughput among all subflows. Since the subflows send the same data and do not impact each other, they behave just like single-path TCP flows. Therefore, we can say that properly sending redundant packets avoids the cases in which MPTCP underperforms single-path TCP and can highly improve the overall throughput of MPTCP in asymmetric network scenarios with packet loss.

One may argue that a scheduler should stop sending packets on the subflows with high RTT or high loss rates instead of sending redundant packets. It can be a good choice if the objective is reducing energy consumption or saving mobile data usage. However, to improve transmission performance, sending redundant packets is a better choice. Firstly, the redundant packets can be used to detect network changes so that MPTCP can react to network changes in time. Specifically, MPTCP collects information about the network paths (e.g., transmission delay or packet loss) by sending packets and receiving acknowledgments. The information is crucial in MPTCP's congestion control and packet scheduling mechanisms. If MPTCP sends no packets or not enough packets to a subflow, there may not be enough information for MPTCP to know about its current network conditions timely, which may result in inappropriate utilization of the network resources. Secondly, redundant packets can improve the robustness of MPTCP transmission: If a packet is lost or delayed, it can be recovered soon with its redundant copies from other subflows. Redundant packets largely shorten the time needed to recover a packet compared with packet retransmissions which need to be triggered with extra communications between hosts.

C. Dealing With Dynamic Networks

In UCB1, we assume that the rewards of a certain arm follow an unknown distribution, but such an assumption may not always hold in dynamic scenarios. Therefore, it becomes a limitation to use the UCB1 algorithm in wireless networks with ever-changing conditions. A fundamental solution to this issue is to remove the assumption of the fixed distribution behind each arm. By doing so, we can formulate the packet scheduling problem as a non-stochastic MAB problem (or adversarial bandit problem).

Unlike the basic MAB problem mentioned above, the non-stochastic MAB problem makes no statistical assumption about the rewards of the slot machines [29]. It thus suits the use cases where the rewards are difficult or impossible to be modeled by an appropriate statistical assumption. Obviously, scheduling packets over multiple dynamic network paths is one of these sophisticated use cases. In this typical scenario, although we can get measurements such as RTT, packet loss rate, and congestion window size of subflows, it is still hard to determine the right statistical assumptions for the relationship between these measurements and the rewards (i.e., the throughput of an MPTCP connection).

Exp3 is an effective solution for the adversarial bandit problem [29]. Generally speaking, Exp3 will still explore other arms even after it has found the best arm that yields the highest rewards currently. Such behavior makes it more suitable for cases where the reward distributions of arms are uncertain (typically in dynamic network scenarios) but also causes shortcomings such as slower convergence and less exploitation of the best arm. Although compared with UCB1, Exp3 seems to be a better algorithm for MPTCP packet scheduling in dynamic mobile networks, its shortcomings reduce the actual transmission performance. Finally, we choose UCB1 instead of Exp3 in OLAPS for two reasons: 1) It makes OLAPS take a shorter time period to learn an optimal policy, which makes it timely determine and fully exploit a proper action. Thus UCB1-based OLAPS can achieve higher throughput than the Exp3-based variant; 2) With the help of the reward monitor, UCB1-based OLAPS can also swiftly switch to a new policy when the network changes, which removes the limitation of the assumption of "fixed distribution for each arm" in a stochastic MAB problem.

We implement both the UCB1-based and Exp3-based OLAPS for performance evaluation in Section VII, and we thoroughly make performance comparisons about the two algorithms in different network scenarios.

D. MPTCP Connections With More Than Two Subflows

The design and evaluations of OLAPS focus on the cases in which each MPTCP connection includes two subflows. That is because OLAPS is designed to be used in today's mobile networks, where devices are most commonly equipped with two network interfaces (i.e., LTE and Wi-Fi).

When there are more than two subflows in a connection, OLAPS can still function properly. However, the scheduling policy derived by the learning agent may be inefficient. For example, when there are two subflows with good network conditions, and another subflow runs on an ill-conditioned network path (e.g., with high delay or packet loss), OLAPS may send more redundant packets to all three subflows. Such a policy prevents the ill-conditioned subflow from hurting the overall performance, but it is inefficient: sending redundant packets on the two "good" subflows is a waste of the goodput.

In a word, ill-conditioned subflows may hurt the overall efficiency. Such an issue can be solved easily with a "patch" as follows:

When there are more than two subflows in an MPTCP connection, OLAPS will exclude the ill-conditioned subflows from the scheduling process. That means OLAPS will not send any packet to these subflows, and the online learning agent will not take them into consideration. This way, the redundant packets that should have been sent to eliminate the negative effects of delayed or lost packets on the ill-conditioned subflows are no longer necessary. Instead, since the subflows still included in the scheduling will be in good network conditions, the agent will learn a policy that sends fewer redundant packets for higher overall throughput and efficiency.

OLAPS can identify the ill-conditioned subflows by leveraging the collected information (e.g., acknowledged and unacknowledged packets, round-trip time, lost packets, etc.). Note that collecting the information is a part of OLAPS in itself (for reward calculation), and the information can be easily collected when sending packets and receiving acknowledgments. Then, when the round-trip time and the packet loss rate of a subflow exceed predefined thresholds (e.g., 150 ms and 0.2), this subflow is considered ill-conditioned. OLAPS then excludes the ill-conditioned subflows from the online learning process until there are no ill-conditioned subflows or only two subflows left.

This patch improves the efficiency in scenarios of more than two subflows without significant modifications to the design of OLAPS.

VII. Evaluation in the Testbed

A. The Semi-Physical Simulation Testbed

In this section, we describe our semi-physical simulation testbed consisting of real hosts and *ns-3* simulated networks. *ns-3* [18] is a popular network simulator that can simulate various kinds of networks in a bottom-up manner. One of its features is the ability to create a simulated network that can be driven by "real" hosts, which provides many conveniences in the evaluation of OLAPS. On the one hand, since the hosts run real Linux kernels, it is much easier and more convincing to compare our scheme with other existing scheduling algorithms over an actual Linux network stack than in a simulated network stack. On the other hand, we can operate repeatable experiments in the simulated networks to draw statistically significant conclusions and avoid the effects of random events, which is nearly impossible to achieve in real networks. Besides, thanks to the powerful features of *ns-3*, we can create networks with different bandwidths, one-way-delay (OWD), and packet loss rate and also simulate the network failures in a real wireless environment, which brings much facticity to the experiments.

A typical topology of the testbed is presented in Fig. 3. To be specific, *ns-3* can handle real network traffic using the *tap-bridge* module, which connects the real Linux network bridges to the tap devices used by *ns-3* scripts. Since the *tap-bridge* module only supports *CSMA* and *Wi-Fi* devices in *ns-3* modules, we add a CSMA channel to connect the two hosts. The CSMA channel has a very high data rate and brings no extra delay, and thus it brings little effect on the overall

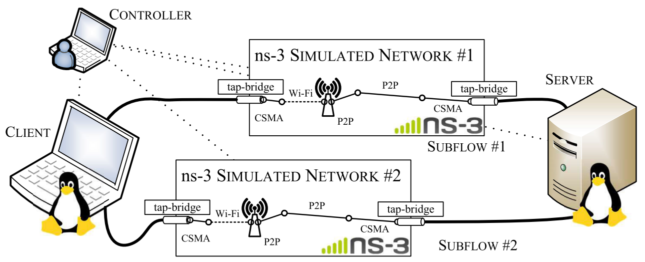


Fig. 3. The topology of the testbed.

performance. Furthermore, we add a *P2P* channel to bring in extra OWD or packet loss to explore the performance in network scenarios with heterogeneous conditions and packet loss. Finally, we use a Wi-Fi channel to connect the two hosts, which is the actual bottleneck of the whole link, to support our experiments in mobile scenarios. Finally, we send *ssh* command from a controller to launch the server, the client, and two *ns-3* scripts simultaneously.

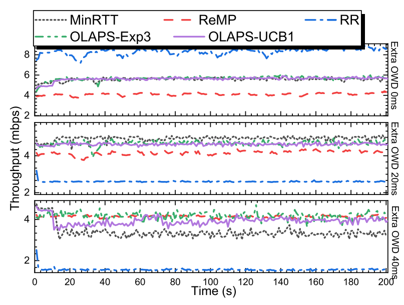
When launching an experiment, the *ns-3* scripts build two simulated networks and the server, and the client establishes an MPTCP connection with two subflows with the *iperf3* [51] application. Even though two simulated networks share the same wireless access components in ns-3, they can be totally different if we tune the ns-3 module with different parameters. In the following parts of this section, we leverage the ns-3 Wi-Fi module and set different one-way-delay (OWD), random loss rate, and user distance to access point (AP) for the two different wireless networks in Fig. 3. Furthermore, we also limit the receiver buffer size to evaluate whether OLAPS works well for different network devices. We give the parameter settings of each set of experiments.

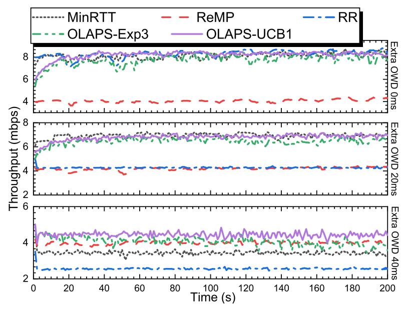
B. OLAPS Performs Consistently Well in Heterogeneous and Lossy Networks

In this section, we compare the performance of UCB1-based and Exp3-based OLAPS (denoted by **OLAPS-UCB1** and **OLAPS-Exp3** respectively), as well as other existing schedulers in the MPTCP Linux kernel, i.e., MinRTT, ReMP, and Round-robin (RR). The experiments include scenarios with homogeneous and heterogeneous network paths, as well as scenarios with or without packet loss. Specifically, we use *ns-3*'s default settings for the 5 GHz Wi-Fi channel with 802.11n standard in the simulated network #1 and #2, and we also introduce extra OWD or packet loss for the P2P channel in the network #2. These experiments aim at exploring the performance of OLAPS and other schedulers and showing how OLAPS provides consistently good performance in almost all scenarios. Collectively, OLAPS-UCB1 and OLAPS-Exp3 are sometimes referred to as **OLAPS variants** in this section.

1) Paths With Different RTTs: We introduce extra OWD in the network #2 and compare OLAPS with other schedulers. The experiment repeats eight times under the same network scenario, and the average throughput of the MPTCP connections per second is presented in Fig. 4. The overall average *throughput gain over ReMP* is shown in Fig. 5. The throughput gain over ReMP of a scheduler means that we divide the throughput of the scheduler by the throughput of ReMP.

The top layer of Fig. 4 shows the instantaneous throughput where the paths are totally symmetric (i.e., no extra OWD nor


(a) Receive buffer size is 30 KB.


(b) Receive buffer size is 50 KB.

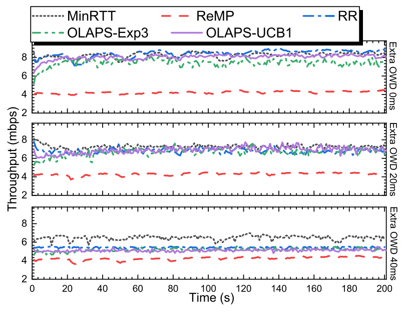

(c) Receive buffer size is 100 KB.

Fig. 4. The instantaneous throughput during the MPTCP connections with different schedulers and receive buffer size.

loss rate is introduced). When the receive buffer is strictly limited to $30KB$ (Fig. 4a), RR outperforms all other schedulers because it evenly utilizes two subflows, and no HoL blocking issue is encountered. Because MinRTT and OLAPS choose to fill the CWND of one specific subflow, they create queuing delay at that subflow's send buffer. That means packets with higher sequence numbers may arrive earlier from another subflow (i.e., HoL blocking happens). The result shows that a small buffer is vulnerable to HoL blocking.

As the receive buffer enlarges, MinRTT catches up with RR in symmetric scenarios and shows its superiority when the networks are asymmetric. However, both of them require larger buffer sizes to maintain their high throughput and underperform ReMP when higher OWD is introduced. As shown in Fig. 4b and Fig. 4c, $50KB$ receive buffer is enough for MinRTT and RR to achieve the highest possible throughput in symmetric networks (2 times of the throughput that ReMP can achieve). However, in asymmetric scenarios (i.e., with an extra $40ms$ OWD), MinRTT and RR suffer severe throughput decline.

In the bottom graph of Fig. 4a and Fig. 4b, neither RR nor MinRTT can keep the throughput improvement feature of multipath transmission, and their throughput is even lower than what the most conservative ReMP provides. Note that in these cases, the performance of ReMP can be considered equivalent to the best performance among all single-path TCP running on the same network paths. This means MPTCP may benefit mobile devices with large buffers or in symmetric network conditions but may sometimes negatively affect the transmission performance of buffer-limited devices in asymmetric networks, which is not uncommon in real mobile networks.

However, both OLAPS-UCB1 and OLAPS-Exp3 can provide consistently good transmission performance no matter how asymmetric the different network paths are and/or how limited the receive buffer is. With the online-learning methods, OLAPS can effectively generate different optimal scheduling policies under different network conditions. Specifically, it tries to aggregate bandwidth from multiple subflows to provide higher throughput as much as possible. The average throughput gains of OLAPS-UCB1 and MinRTT in homogeneous network scenarios in Fig. 5 are very similar, i.e., $1.75\times$ and $1.77\times$, respectively. Moreover, it will fall back to a conservative policy to send redundant packets when it notices that the policy of aggregating bandwidth in such a harsh network scenario may eventually lead to a lower throughput compared with a single TCP flow. As shown in Fig. 5, MinRTT and RR underperform the ReMP scheduler in heterogeneous

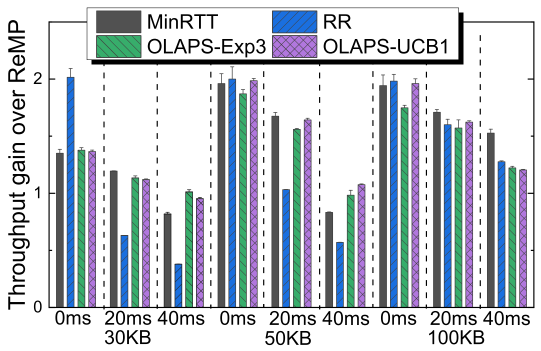
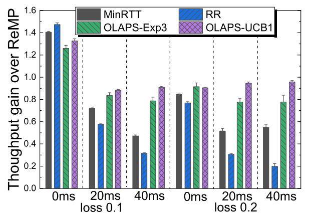


Fig. 5. The average throughput gain over ReMP.


(a) Receive buffer size is 50 KB.

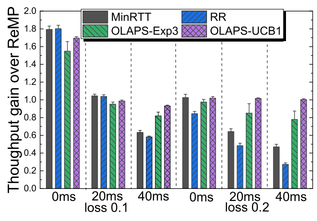

(b) Receive buffer size is 100 KB.

Fig. 6. The average throughput gain over ReMP with heterogeneous network conditions and packet loss.

network scenarios, but OLAPS never does. In the cases where MinRTT suffers heterogeneous network paths, OLAPS-UCB1 eliminates dramatic throughput degradation and provides up to $1.29\times$ the throughput of MinRTT. This result illustrates that there is no need to worry that MPTCP with OLAPS will underperform single-path TCP, which can be a significant motivation for deploying MPTCP.

It is worth mentioning that, in some cases, OLAPS does not provide the highest throughput. That is because the online-learning agent needs sufficient time to learn the optimal policy and will explore the actions that may lead to a suboptimal transmission performance.

OLAPS-UCB1 outperforms OLAPS-Exp3 in having a shorter convergence time and achieving up to $1.12\times$ the throughput. The reason is that the Exp3 algorithm will always explore the suboptimal actions even when there is one action that provides the highest throughput. Meanwhile, UCB1 always sticks to such action after it has done enough explorations. But when several actions yield similar rewards, UCB1 will explore more suboptimal actions to find out the best action finally.

2) Lossy Scenarios: We also conduct experiments to explore different performances of OLAPS with packet loss by introducing a random packet loss rate in the P2P channel of network #2. We repeat eight runs for each scheduler in each scenario, and the results are shown in Fig. 6. Likewise, we present the *throughput gain over ReMP* instead of the actual value of throughput for easy reading. Specifically, the average throughput of ReMP shown in Fig. 6a and Fig. 6b are $4.24\ mbps$ and $4.02\ mbps$, respectively.

Fig. 6 illustrates that MPTCP with OLAPS is still a safe bet when random packet loss happens. That is, when sending non-duplicate packets on paths is an effective strategy, OLAPS can learn such a strategy and achieve higher throughput than ReMP. And when the lost and delayed packets on a subflow cause a dramatic decline in the transmission performance, OLAPS can learn a conservative policy that sends packets redundantly on both paths. Since the losses of redundant packets on a subflow will not affect the overall transmission, OLAPS can still achieve the same throughput as ReMP. On the contrary, MinRTT and RR suffer a great decline in throughput. Though they can achieve higher throughput than OLAPS when no extra OWD is introduced, their throughput dramatically drops to less than 0.5 times compared with what ReMP provides with $40ms$ OWD and 0.2 random loss rate.

The results show that OLAPS-UCB1 is the best choice from a comprehensive perspective. In Fig. 6a, OLAPS-UCB1 outperforms MinRTT and RR by achieving $1.46\times$ and $2.40\times$ more throughput, respectively, on average of all network scenarios in the evaluations. Though ReMP achieves the highest average throughput in this case, OLAPS-UCB1 achieves a very close 98.7% of ReMP's average throughput. In Fig. 6b, OLAPS-UCB1 outperforms MinRTT and RR by achieving $1.34\times$ and $1.75\times$ more throughput, respectively, and achieves the highest average throughput, which is $1.11\times$ higher than second place ReMP.

Comparing the two variants of OLAPS, OLAPS-UCB1 outperforms OLAPS-Exp3 in most cases. When a certain action yields a much higher average reward than others, UCB1 can fully exploit such an action. Exp3, however, will keep exploring other actions since it assumes the reward distributions may change anytime. Averagely, OLAPS-UCB1 provides $1.12\times$ and $1.13\times$ throughput than OLAPS-Exp3 as shown in Fig. 6a and Fig. 6b, respectively.

In general, the two OLAPS variants may not be able to achieve the highest throughput all the time as they need to explore the actions that may lead to suboptimal transmission performance. However, in exchange, OLAPS holds high adaptability to various network scenarios and can take proper actions with a high probability, which makes OLAPS the best choice.

C. Fast Convergence in Dynamic Networks

Abrupt network change and user mobility are common in mobile networks, and an important criterion for learning-based methods is the convergence time to switch to a stable new policy when the environment changes. As a result, we conduct experiments to evaluate our OLAPS variants in changing environments. Since the reward monitor in OLAPS is supposed to shorten the convergence time of OLAPS, we compare the performances of OLAPS-UCB1 and OLAPS-Exp3 with or without the reward monitor. "OLAPS-UCB1 w/o Monitor" and "OLAPS-Exp3 w/o Monitor" denote the related OLAPS variants without the reward monitor.

We design two kinds of *ns-3* networks to simulate the network change and user mobility scenarios, respectively.

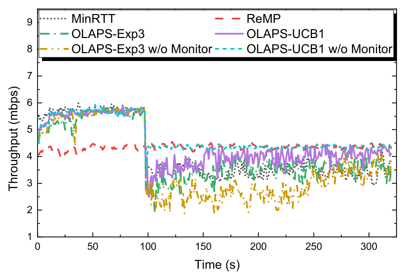
1) Abrupt Network Change: In this experiment, we launch an MPTCP connection for 320 seconds and change the network conditions after 100 seconds. All the experiments start with no extra OWD nor random loss rate. Then after 100 seconds, we introduce an extra $30ms$ OWD in network #2 for the first set of experiments and 0.1 loss rate for the second set. We run eight times and present the average throughput of the MPTCP connections at every second in Fig. 7.

The result shows that although both OLAPS variants can achieve high throughput in the first 100 seconds with or without the reward monitor, their behaviors are different when network changes lead to changes in optimal scheduling policy.

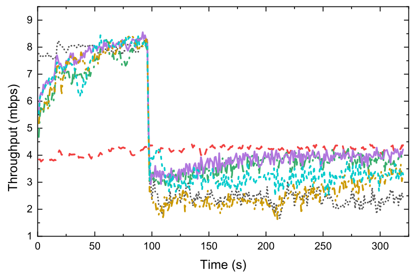
In general, OLAPS-UCB1 outperforms OLAPS-Exp3 in that it has a shorter convergence time and further achieves a higher throughput, as shown in Fig. 7. And further performance evaluation results verify the necessity of introducing the reward monitor. Fig. 7b and Fig. 7c clearly illustrate that without the reward monitor, the UCB1 algorithm can easily lose its ability to learn an optimal policy when the network changes, i.e., OLAPS-UCB1 w/o Monitor can hardly learn to send more redundant packets in lossy networks. On the one hand, the policy of sending fewer redundant packets yields such high rewards in the first 100 seconds, but the lower current rewards after 100 seconds have a small impact on its high average reward. And since UCB1 prefers policies with higher average rewards, it still chooses to send fewer redundant packets even after the network changes, which leads to poor transmission performance. On the other hand, OLAPS-UCB1, with the reward monitor, can precisely detect the network changes and will restart the learning process to learn the optimal policy after the network changes.

However, there is an exception in Fig. 7a. It seems that OLAPS-UCB1 w/o Monitor magically learns the best policy right after the network changes and immediately chooses to use a high "redundant ratio" in the lossy network scenario. Such behavior is a "lucky" mistake for OLAPS-UCB1 w/o Monitor: It chooses a policy in the wrong way, but this policy happens to be the current optimal solution. To be specific, before the network changes, using a lower redundant ratio is the optimal policy, but the gaps between its average reward and that of others are not significant due to the network conditions. After the network changes, the average rewards for using higher redundant ratios get higher. And since these actions are previously not optimal, they are taken less often, which leads to their higher upper confidence bounds according to the UCB1 algorithm. The higher average rewards and upper confidence bounds make using higher redundant ratios the preferred policy of the UCB1 algorithm, which luckily leads to optimal performance. In a word, OLAPS-UCB1 w/o Monitor still cannot learn a new policy after network changes in the right way. It may sometimes be lucky, like in the case in Fig. 7a, but generally, OLAPS-UCB1 with a reward monitor is still the best choice in our work.

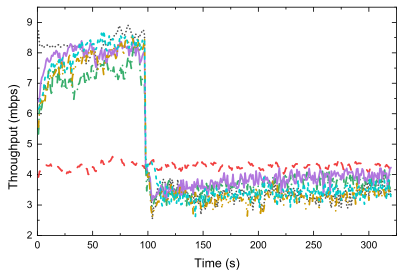
Based on the assumption that there is no fixed reward distribution, OLAPS-Exp3 always has the ability to learn a new optimal policy. But OLAPS-Exp3 w/o monitor needs to spend



(a) Receive buffer size is 30 KB.



(b) Receive buffer size is 50 KB.



(c) Receive buffer size is 100 KB.

Fig. 7. The instantaneous throughput when experiencing abrupt network changes.

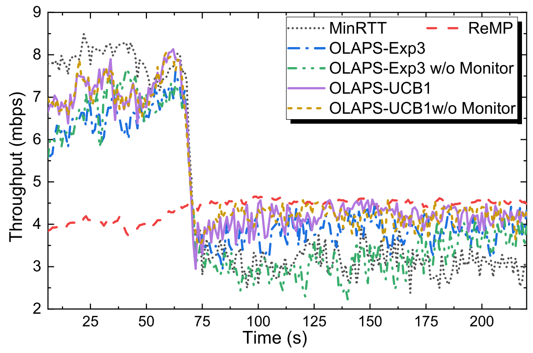


Fig. 8. The instantaneous throughput in mobile scenario.

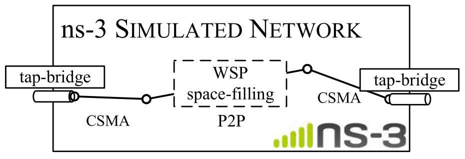


Fig. 9. Another simulated network for factor space exploration.

TABLE II
DOMAINS OF THE INFLUENCING FACTORS

Factors	Path #1	Path #2
Link Capacity (Mbps)	8 − 16	8 − 16
One-Way-Delay (ms)	0 − 5	0 − 40
Loss Rate	0	0 − 0.2
Receive Buffer Size	*Small* $16KB - 512KB$	*Large* $16KB - 1MB$

much more time to reach convergence; thus, it doesn't perform well enough, as shown in Fig. 7. OLAPS-Exp3 w/o Monitor suffers a longer convergence time after the network changes, as the Exp3 agent is unaware of the change and will still choose to use a policy according to its outdated knowledge. It will explore other actions randomly with a small probability since these actions are considered "suboptimal" before the network changes. However, with the reward monitor, OLAPS-Exp3 is able to detect the changes in the network. Thus the Exp3 agent will start to explore other actions with a higher probability, which finally leads to a shorter convergence time to learn a proper scheduling policy.

2) User Mobility: In this experiment, we use the mobility module of *ns-3* to simulate the scenario that the client moves out of the signal range of the Wi-Fi AP (access point) in simulated network #2. As the client moves away from the Wi-Fi AP, the performance of the Wi-Fi channel will decrease over time, leading to a network failure [52]. However, this switch is not necessarily instantaneous. Specifically, the distance between AP and the client changes from 0 meters to 150 meters at a speed of $2m/s$ and then stays at 150 meters. Besides, an extra $10ms$ OWD is introduced in network #2, and the receive buffer size is $100KB$. We run eight times and present the average throughput of the MPTCP connections at every second in Fig. 8.

OLAPS variants show their adaptability to the changing environment. They can learn an optimal policy before or after the client moves out of the signal range. The result illustrates that compared with heuristic schedulers (ReMP and MinRTT), using OLAPS variants could be a better choice for MPTCP in mobile networks. However, OLAPS-Exp3 w/o Monitor suffers a much longer convergence time because, without the reward monitor, the Exp3 agent will stick to a suboptimal action and do a little exploration.

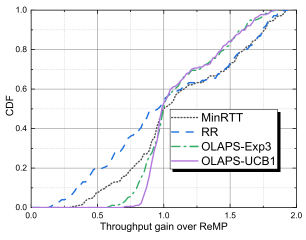
In this case, OLPAS-UCB1 also outperforms OLAPS-Exp3. Before 75 seconds, OLAPS-UCB1 spends less time learning the optimal policy and reaches higher throughput. And after 75 seconds, with the help of the reward monitor, OLAPS-UCB1 quickly learns a new policy, while OLAPS-Exp3 needs more time to converge.

D. Evaluation in the Whole Factor Space

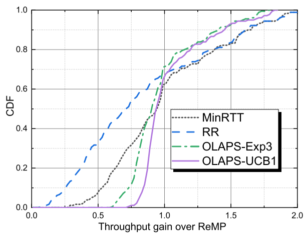
We now evaluate two variants of OLAPS in some typical scenarios to illustrate how its components work. But in real mobile networks, many factors can influence the performance of an MPTCP connection, and their combinations are sophisticated. As a result, in the following experiments, we conduct experiments that run over network paths with random network characteristics that are distributed in a *factor space*. Leveraging the WSP [53] algorithm, we generate around 180 samples spreading over the factor spaces in Table II. That means the schedulers are evaluated under random network conditions, which can represent the schedulers' performances under a wide range of network scenarios. Table II shows the range of the specific network characteristics. For example, the link capacity of both paths can vary from $8-16Mbps$. Especially there are two ranges for the receive buffer size to show the performances of the schedulers with different kinds of user devices.

These settings are applied in the P2P channel of another *ns-3*-based simulated network shown in Fig. 9. We launch an MPTCP connection with each scheduler for 100 seconds and present the cumulative distribution function (CDF) of the throughput gain over ReMP in Fig. 10.

The figures in Fig. 10 illustrate that the most remarkable enhancement of OLAPS is the throughput improvement in those "bad" network scenarios where other schedulers suffer



(a) Large buffer size.



(b) Small buffer size.

Fig. 10. The performance of the schedulers in different scenarios.

TABLE III
AVERAGE THROUGHPUT GAIN IN FIG. 10

Receive Buffer Size	MinRTT	RR	OLAPS-Exp3	OLAPS-UCB1
Large	1.13	1.03	1.12	1.14
Small	1.00	0.84	0.98	1.04

dramatic declines in performance. The figures show that with the randomly chosen network characteristics, there are always some cases in that other schedulers fail to achieve higher throughput than that ReMP can reach. For example, Fig. 10a shows that in 50% of all cases, the throughput gains of all the schedulers are less than 1. In such cases, the throughput gains of MinRTT (RR) are sometimes less than 0.75 in around 20% (35%) of all experiments. However, OLAPS-UCB1 can make sure that the throughput gain is higher than 0.75 in 99% of the experiments. Similar results can be drawn from Fig. 10b, and in this figure, the superiority of OLAPS is magnified since, in more cases, MinRTT and RR suffer declined throughput due to limited buffer size.

On the other hand, Fig. 10a and Fig. 10b also illustrate that OLAPS can not only provide stable transmission performances with harsh network conditions but can also aggregate bandwidth of the two paths and achieve higher throughput than ReMP when possible. Specifically, the throughput gains of OLAPS variants can reach up to 1.8 in both figures, which means OLAPS reaches 1.8 times higher throughput than ReMP in this case. Note that MinRTT and RR can reach higher throughput when the network conditions are relatively good, which is in line with our expectations. OLAPS, as an online-learning-based scheduler, always needs time to learn and explore a scheduling policy. The learning process brings OLAPS optimal scheduling policy with high adaptability to different networks at the cost of some throughput loss when exploring suboptimal actions. However, Table III shows that the throughput loss due to online learning is affordable since the average throughput gain of OLAPS-UCB1 is higher than that of MinRTT and RR.

In conclusion, our experiments show that OLAPS can be an adequate packet scheduling algorithm that makes MPTCP really feasible in mobile networks with complex scenarios. On the one hand, when the network conditions of subflows are good enough, OLAPS aggregates the bandwidth of different network paths and provides higher throughput than single-path TCP (or MPTCP with ReMP scheduler) can provide. On the other hand, in heterogeneous network scenarios and the scenario with packet loss, which can never be ignored in real networks, OLAPS avoids the dramatic decline in throughput that haunts the in-kernel schedulers and can still provide acceptable throughput. Besides, the online-learning agent of OLAPS can always learn a proper policy that suits current network conditions, and OLAPS further accelerates the learning process.

VIII. LIMITATIONS AND FUTURE WORK

OLAPS uses an online-learning process to learn a better packet scheduling policy. Though it is a deployable lightweight solution that needs no training data, it takes time before a proper policy is learned. When the mobile network changes faster than OLAPS can learn a stable policy, it can only behave conservatively to provide a throughput similar to that a single-path TCP can provide. Although such a performance has already been better than other in-kernel schedulers can provide under the same circumstance, faster reaction and higher throughput are always desirable. For future work, we consider combining OLAPS with an offline-learning process to train a helpful neural network. Besides, how to make an offline-learning-based design deployable is also a challenging task to solve.

Another part of OLAPS that needs improvement is its ability to achieve higher throughput in heterogeneous network scenarios with packet loss. The current design of OLAPS uses a simple yet effective way to choose the next subflow, which is to choose the subflow with the lowest RTT. This design works well in a homogeneous network, but when network paths differ a lot, it may cause HoL blocking, which drives the online-learning agent to play a conservative policy by sending more redundant packets. If the effectiveness of the algorithm in choosing a subflow can be improved, higher throughput can be achieved. However, researchers have proven that many state-of-art schedulers actually behave unsatisfactorily in such scenarios [25]. As a result, we are planning to design such an algorithm to improve the overall performance of OLAPS.

Furthermore, we are interested in making OLAPS aware of different quality-of-service (QoS) requirements from various applications. For example, video streaming applications may not care about the overall throughput that much. Instead, they require low application delay. OLAPS's pursuit of instantaneous throughput should benefit such applications more than those pursuing overall throughput. However, it does not include an explicit design to reduce transmission delay or fulfill other QoS requirements. Since such a design needs cross-layer communication, which means it is not transparent to upper layers, we leave it for future work and make sure the deployment of the current OLAPS needs no extra modification on both the server and client sides.

IX. CONCLUSION

MPTCP is a promising protocol that enables the concurrent use of multiple network paths for higher throughput and better resilience to network failures. However, in complex mobile networks, MPTCP may even underperform single-path TCP, and one of the main reasons is that existing packet scheduling algorithms in the MPTCP Linux kernel may work well in some cases but perform awfully in others. A truly deployable algorithm may not perform the best, but it should consistently provide good performances in most cases.

With that purpose, we propose OLAPS, a multipath packet scheduler assisted by an online-learning agent, to use adaptive scheduling policies to suit different network conditions.

OLAPS needs not only to provide high aggregated throughput when network conditions are relatively homogeneous and loss-free but also to improve MPTCP's performance in harsh network scenarios with heterogeneous paths, random packet loss, and buffer-limited receiving devices. Our experiments show that OLAPS achieves its goals through its well-designed online-learning agent, whose learning process is further accelerated by a reward monitor. Since OLAPS is a lightweight algorithm needing no training data and can work as a Linux kernel module without any modification needed on the client side, it can serve as one of the packet scheduling algorithms and promote the deployment of MPTCP in practical use.

REFERENCES

[1] *Cisco Annual Internet Report (2018–2023) White Paper*. Accessed: Feb. 2023. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] Y. Li et al., "A nationwide study on cellular reliability: Measurement, analysis, and enhancements," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2021, pp. 597–609.

[3] J. R. Iyengar, P. D. Amer, and R. Stewart, "Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths," *IEEE/ACM Trans. Netw.*, vol. 14, no. 5, pp. 951–964, Oct. 2006.

[4] Q. De Coninck and O. Bonaventure, "Multipath QUIC: Design and evaluation," in *Proc. 13th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2017, pp. 160–166.

[5] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, and C. Paasch, *TCP Extensions for Multipath Operation With Multiple Addresses*, document RFC 8684, 2020. Accessed: Feb. 2023. [Online]. Available: https://www.ietf.org/rfc/rfc8684.txt

[6] O. Bonaventure. *Apple Music on iOS13 Uses Multipath TCP Through Load-Balancers*. Accessed: Feb. 2023. [Online]. Available: http://blog.multipath-tcp.org/blog/html/2019/10/27/apple_music_on_ios13_uses_multipath_tcp_through_load_balancers.html

[7] Apple. *Use Multipath TCP to Create Backup Connections for iOS*. Accessed: Feb. 2023. [Online]. Available: https://support.apple.com/en-us/HT201373

[8] O. Bonaventure and S. Seo, "Multipath TCP deployments," *IETF J.*, vol. 12, no. 2, pp. 24–27, Nov. 2016.

[9] C. Paasch and O. Bonaventure, "Multipath TCP," *Commun. ACM*, vol. 57, no. 4, pp. 51–57, Apr. 2014.

[10] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM Conf. (SIGCOMM)*, 2011, pp. 266–277.

[11] L. Li et al., "A measurement study on multi-path TCP with multiple cellular carriers on high speed rails," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2018, pp. 161–175.

[12] C. Paasch and S. Barre. *Multipath TCP in the Linux Kernel*. Accessed: Feb. 2023. [Online]. Available: https://www.multipath-tcp.org

[13] A. Frommgen, T. Erbshauser, A. Buchmann, T. Zimmermann, and K. Wehrle, "ReMP TCP: Low latency multipath TCP," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2016, pp. 1–7.

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

[15] Y.-S. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "ECF: An MPTCP path scheduler to manage heterogeneous paths," in *Proc. 13th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2017, pp. 147–159.

[16] B. Arzani, A. Gurney, S. Cheng, R. Guerin, and B. T. Loo, "Impact of path characteristics and scheduling policies on MPTCP performance," in *Proc. 28th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, May 2014, pp. 743–748.

[17] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, 2002.

[18] *A Discrete-Event Network Simulator for Internet Systems*. Accessed: Feb. 2023. [Online]. Available: https://www.nsnam.org/

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

[20] R. Khalili, N. Gast, M. Popovic, and J.-Y. Le Boudec, "MPTCP is not Pareto-optimal: Performance issues and a possible solution," *IEEE/ACM Trans. Netw.*, vol. 21, no. 5, pp. 1651–1665, Oct. 2013.

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

[22] B. Partov and D. J. Leith, "Experimental evaluation of multi-path schedulers for LTE/Wi-Fi devices," in *Proc. 10th ACM Int. Workshop Wireless Netw. Testbeds, Exp. Eval., Characterization*, Oct. 2016, pp. 41–48.

[23] O. Bonaventure, C. Paasch, and G. Detal, *Use Cases and Operational Experience With Multipath TCP*, document RFC 8041, 2020. Accessed: Feb. 2023. [Online]. Available: https://www.ietf.org/rfc/rfc8041.txt

[24] F. Yang, Q. Wang, and P. D. Amer, "Out-of-order transmission for in-order arrival scheduling for multipath TCP," in *Proc. 28th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, May 2014, pp. 749–752.

[25] P. Hurtig, K.-J. Grinnemo, A. Brunstrom, S. Ferlin, O. Alay, and N. Kuhn, "Low-latency scheduling in MPTCP," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 302–315, Feb. 2019.

[26] H. Zhang, W. Li, S. Gao, X. Wang, and B. Ye, "ReLeS: A neural adaptive multipath scheduler based on deep reinforcement learning," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2019, pp. 1648–1656.

[27] K. Yedugundla et al., "Is multi-path transport suitable for latency sensitive traffic?" *Comput. Netw.*, vol. 105, pp. 1–21, Aug. 2016.

[28] H. Wu, O. Alay, A. Brunstrom, S. Ferlin, and G. Caso, "Peekaboo: Learning-based multipath scheduling for dynamic heterogeneous environments," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2295–2310, Oct. 2020.

[29] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multiarmed bandit problem," *SIAM J. Comput.*, vol. 32, no. 1, pp. 48–77, 2011.

[30] T. L. Lai and H. Robbins, "Asymptotically efficient adaptive allocation rules," *Adv. Appl. Math.*, vol. 6, no. 1, pp. 4–22, Mar. 1985.

[31] H. Robbins, "Some aspects of the sequential design of experiments," *Bull. Amer. Math. Soc.*, vol. 58, no. 5, pp. 527–535, 1952.

[32] G. Sarwar, R. Boreli, E. Lochin, A. Mifdaoui, and G. Smith, "Mitigating receiver's buffer blocking by delay aware packet scheduling in multipath data transfer," in *Proc. 27th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, Mar. 2013, pp. 1119–1124.

[33] K. Xue et al., "DPSAF: Forward prediction based dynamic packet scheduling and adjusting with feedback for multipath TCP in lossy heterogeneous networks," *IEEE Trans. Veh. Technol.*, vol. 67, no. 2, pp. 1521–1534, Feb. 2018.

[34] 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 Trans. Veh. Technol.*, vol. 70, no. 1, pp. 914–927, Jan. 2021.

[35] H. Lee, J. Flinn, and B. Tonshal, "RAVEN: Improving interactive latency for the connected car," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2018, pp. 557–572.

[36] Z. Zheng et al., "XLINK: QoE-driven multi-path QUIC transport in large-scale video services," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2021, pp. 418–432.

[37] S. R. Pokhrel and M. Mandjes, "Improving multipath TCP performance over WiFi and cellular networks: An analytical approach," *IEEE Trans. Mobile Comput.*, vol. 18, no. 11, pp. 2562–2576, Nov. 2019.

[38] S. Raj Pokhrel and C. Williamson, "A rent-seeking framework for multipath TCP," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 48, no. 3, pp. 63–70, Mar. 2021.

[39] S. R. Pokhrel and S. Garg, "Multipath communication with deep Q-network for industry 4.0 automation and orchestration," *IEEE Trans. Ind. Informat.*, vol. 17, no. 4, pp. 2852–2859, Apr. 2021.

[40] S. R. Pokhrel and J. Choi, "Improving TCP performance over WiFi for internet of vehicles: A federated learning approach," *IEEE Trans. Veh. Technol.*, vol. 69, no. 6, pp. 6798–6802, Jun. 2020.

[41] S. Abbasloo, C.-Y. Yen, and H. J. Chao, "Wanna make your TCP scheme great for cellular networks? Let machines do it for you!" *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 265–279, Jan. 2021.

[42] C. Huang, J. Zhang, and T. Huang, "Objective-oriented resource pooling in MPTCP: A deep reinforcement learning approach," in *Proc. 3rd Int. Conf. Hot Inf.-Centric Netw. (HotICN)*, Dec. 2020, pp. 175–181.

[43] J. Chung, D. Han, J. Kim, and C.-K. Kim, "Machine learning based path management for mobile devices over MPTCP," in *Proc. IEEE Int. Conf. Big Data Smart Comput. (BigComp)*, Feb. 2017, pp. 206–209.

[44] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2017, pp. 3389–3396.

[45] T. Lu, D. Pal, and M. Pal, "Contextual multi-armed bandits," in *Proc. 13th Int. Conf. Artif. Intell. Statist. (AISTATS)*, vol. 9, 2010, pp. 485–492.

[46] L. Li, W. Chu, J. Langford, and R. E. Schapire, "A contextual-bandit approach to personalized news article recommendation," in *Proc. 19th Int. Conf. World wide web*, Apr. 2010, pp. 661–670.

[47] D. Chandramouli and T. Sun, *System Architecture for the 5G System (5GS)*, document TS 23.501, Version 17.5.0, Release 17, 3rd Generation Partnership Project (3GPP), 2022.

[48] C. Paasch, "Improving multipath TCP," Ph.D. dissertation, Louvain School Eng., Université Catholique de Louvain, Ottignies-Louvain-la-Neuve, Belgium, 2014.

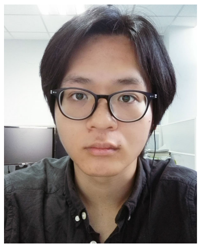
[49] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson, "BBR: Congestion-based congestion control," *Commun. ACM*, vol. 60, no. 2, pp. 58–66, 2017.

[50] M. Dong et al., "PCC vivace: Online-learning congestion control," in *Proc. 15th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2018, pp. 343–356.

[51] J. Dugan, S. Elliott, B. A. Mah, J. Poskanzer, and K. Prabhu. *iPerf—The Ultimate Speed Test Tool for TCP, UDP and SCTP*. Accessed: Feb. 2023. [Online]. Available: https://iperf.fr/

[52] Q. De Coninck and O. Bonaventure, "MultipathTester: Comparing MPTCP and MPQUIC in mobile environments," in *Proc. Netw. Traffic Meas. Anal. Conf. (TMA)*, Jun. 2019, pp. 221–226.

[53] J. Santiago, M. Claeys-Bruno, and M. Sergent, "Construction of space-filling designs using WSP algorithm for high dimensional spaces," *Chemometric Intell. Lab. Syst.*, vol. 113, pp. 26–31, Apr. 2012.


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

Kaiping Xue (Senior Member, IEEE) received the bachelor's degree from the Department of Information Security, University of Science and Technology of China (USTC), in 2003, and the Ph.D. degree from the Department of Electronic Engineering and Information Science (EEIS), USTC, in 2007. From May 2012 to May 2013, he was a Post-Doctoral Researcher with the Department of Electrical and Computer Engineering, University of Florida. He is currently a Professor with the School of Cyber Science and Technology, USTC. His research interests include next-generation internet architecture design, transmission optimization, and network security. He is an IET Fellow. He serves on the Editorial Board of several journals, including the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING (TDSC), the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS (TWC), and the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT (TNSM). He has also served as the (Lead) Guest Editor for many reputed journals/magazines, including IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (JSAC), *IEEE Communications Magazine*, and IEEE NETWORK.

Yuan Zhang received the bachelor's and master's degrees from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 2019 and 2022, respectively. His research interests include future internet architecture design and transmission optimization.

Jiangping Han (Member, IEEE) received the bachelor's and Ph.D. degrees from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 2016 and 2021, respectively. She was a Visiting Student at Arizona State University from 2020 to 2021. She is currently a Post-Doctoral Fellow with the School of Cyber Science and Technology, USTC. Her research interests include data center networks, future internet architecture design, and transmission optimization.

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

David S. L. Wei (Life Senior Member, IEEE) received the Ph.D. degree in computer and information science from the University of Pennsylvania in 1991. From May 1993 to August 1997, he was on the Faculty of Computer Science and Engineering, The University of Aizu, Japan, as an Associate Professor and then a Professor. He is currently a Professor with the Computer and Information Science Department, Fordham University. He has authored and coauthored more than 140 technical papers in various archival journals and conference proceedings. His research interests include cloud and edge computing, cybersecurity, and quantum computing and communications. He is a member of ACM and AAAS and a Life Senior Member of the IEEE Computer Society and the IEEE Communications Society. He was a recipient of the IEEE Region 1 Technological Innovation Award (Academic) in 2020, for contributions to information security in wireless and satellite communications and cyber-physical systems. He was a Lead Guest Editor or the Guest Editor of several special issues in the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, the IEEE TRANSACTIONS ON CLOUD COMPUTING, and the IEEE TRANSACTIONS ON BIG DATA. He also served as an Associate Editor for IEEE TRANSACTIONS ON CLOUD COMPUTING from 2014 to 2018 and *Journal of Circuits, Systems and Computers* from 2013 to 2018 and an Editor for IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (JSAC) for the Series on Network Softwarization and Enablers from 2018 to 2020.