A Low-Latency MPTCP Scheduler for Live Video Streaming in Mobile Networks

Yitao Xing, Kaiping Xue, *Senior Member, IEEE*, Yuan Zhang,
Jiangping Han, *Graduate Student Member, IEEE*, Jian Li, *Member, IEEE*,
Jianqing Liu, *Member, IEEE*, and Ruidong Li, *Senior Member, IEEE*

***Abstract*— It is a known issue that low-latency communication is hard to achieve when using multiple network interfaces with asymmetric capacity and delay (e.g., LTE and WLAN) simultaneously. A main underlying cause of this issue is that the packets with lower sequence number are stalled on a high-latency path, thus the early arriving packets with higher sequence number become "out-of-order (OFO)" packets. These OFO packets may excessively consume receiver's buffer, causing long reordering delay and unnecessary packet retransmission. In this paper, we present a novel design of packet scheduling for Multipath TCP (MPTCP), called OverLapped Scheduler (OLS), able to tackle the OFO-packet problem more effectively. OLS can guarantee sufficient throughput on demand of upper layer applications, and utilizes the remaining bandwidth to reduce OFO-packets. To do so, OLS schedules packets according to their arrival time and sends a controlled number of redundant packets to avoid the impact of inaccurate arrival-time estimations due to network jitter. We implement OLS in a Linux kernel, and the experiments show that in asymmetric networks with or without jitter, OLS can effectively reduce OFO-packets and transmission latency while maintaining a sufficient throughput, which makes it fully capable to meet the requirements of applications such as live video streaming.**

***Index Terms*— Multipath TCP, live video streaming, out-of-order packets, scheduler.**

I. Introduction

LIVE video streaming applications are getting a worldwide popularity, and with the increasing number of mobile users and devices, providing proper transmission service for live streaming applications in mobile networks has become an urgent issue. According to [1], global mobile data traffic will grow sixfold from 2017 to 2022, and Internet video will represent 82% of all business Internet traffic by 2022. Since mobile users are expecting to get access to the internet anytime anywhere, mobile devices nowadays are usually equipped with multiple network interfaces to work with various access technologies such as 4G, 5G, Wi-Fi and Wi-Fi6. As a result, live video streaming traffic may encounter different network conditions, which challenges the data transmission underlying.

To be more specific, live streaming applications have high requirements on both bandwidth and transmission latency. On the one hand, live streaming is bandwidth-consuming with the need of high-quality video streaming capabilities. For example, there is a trend that Ultra-High-Definition (UHD) and 4K video streaming is becoming prevalent [1], and Multi-View Video (MVV) as well as 8K will also be in the foreseeable future, which will consume much more amount of network bandwidth. On the other hand, some video streaming applications nowadays require low and even near real-time transmission latency. Such applications include two-way chat, video conferencing, real-time device control, cloud gaming, etc., which are springing up rapidly in recent years.

Providing data transmission service for live streaming applications with such complex requirements is quite challenging, especially in mobile networks. For one thing, in wireless environments where bandwidth is scarce and expensive, to expand the bandwidth capacity of a single network interface needs long-term research and huge investment(e.g., the emerging 5G standard). For another, to provide low-latency and robust data transmission is also a tough task because of user mobility, limited signal range, frequent handoffs and other common issues in mobile networks. To tackle these problems, a feasible approach is to utilize multiple network interfaces equipped in almost every mobile device simultaneously, such as LTE and WLAN access. Such design is deemed as a multipath protocol.

Several developed multipath protocols have shown their potential for providing higher aggregated bandwidth, improved robustness and better support for user mobility compared with single-path protocols. For example, there are some transport layer protocols such as CMT-SCTP [2], multipath QUIC (MPQUIC) [3] and multipath TCP (MPTCP) [4], have been widely studied. Among them, MPTCP, as a drop-in extension

Manuscript received January 22, 2021; revised March 30, 2021; accepted May 8, 2021. Date of publication May 25, 2021; date of current version November 11, 2021. 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 Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant JP19H04105. The associate editor coordinating the review of this article and approving it for publication was H. Zhu. *(Corresponding author: Kaiping Xue.)*

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

Yuan Zhang and Jiangping Han are with the Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei 230027, China.

Jianqing Liu is with the Department of Electrical and Computer Engineering, The University of Alabama in Huntsville, Huntsville, AL 35899 USA.

Ruidong Li is with the College of Science and Engineering, Kanazawa University, Kanazawa 920-1192, Japan.

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TWC.2021.3081498.

Digital Object Identifier 10.1109/TWC.2021.3081498

to regular TCP, is now a recent IETF standard [4] and used by major software vendors, including the Siri service of Apple [5], [6]. Besides, MPTCP has also become a part of the ATSSS (Access Traffic Steering, Switching and Splitting) function in 5GC (5G Core) network [7]. MPTCP gains its wide popularity by offering performance enhancements such as seamless handover between network interfaces to support user mobility, resilience to link failures and aggregated bandwidth from multiple paths [8]–[11].

However, using MPTCP straightforwardly does not always guarantee satisfying transmission performance, especially when asymmetric link technologies are used (e.g. LTE and WLAN). In this situation, it is hard for MPTCP to provide low-latency transmission service for delay-sensitive applications, and the underlying cause of this issue is the problem of out-of-order (OFO) packets. Specifically, when data packets do not arrive in original preserved order, packets with higher sequence number have to be stored and remain unprocessed in the receiver's TCP buffer till the reception of packets with lower sequence numbers, which causes occupied (thus wasted) buffer space, head-of-line (HOL) blocking, unnecessary packet retransmissions, and long reordering delay [12]. That makes OFO-packet problem a thorny obstacle for MPTCP to provide low-latency transmission service [13]–[15].

To make MPTCP able to provide low-latency transmission, some scheduling methods have been made to reduce OFO-packets caused by asymmetric network paths. The basic idea behind them is to estimate the arrival time of packets based on network conditions, and then to send packets according to the arrival-time estimations instead of their native sequence numbers [16]–[18]. But OFO-packet problem still remains in practical use, because existing schedulers may not work well in mobile networks with jitter. And the unpredictable network jitter leads to inaccurate arrival-time estimations, which make some well-designed schedulers even underperform the naive Round-Robin scheduler [19] in certain situations.

An alternative approach to getting around the OFO-packet problem is to allocate a larger TCP buffer at receiver, but unfortunately, delay-sensitive applications such as live video streaming is unable to take advantage of it. Though a large enough TCP buffer stores OFO-packets for reordering and the retransmission problem can be mitigated, the HoL blocking and long reordering delay still exist, which delay-sensitive applications such as live streaming care more about. Worse still, mobile devices are usually equipped with limited buffer. As a result, the OFO-packet problem is still a burning issue for delay-sensitive applications and MPTCP should be improved to tackle it.

In this paper, we propose a novel MPTCP scheduling method named OverLapped Scheduler (OLS) to solve the problem, which is designed to trade part of the aggregated bandwidth for fewer OFO-packets in asymmetric networks with jitter. Its design is based on an observation that live video streaming applications always generates traffic with a limited video bitrate, so the aggregated bandwidth higher than the bitrate is idle. OLS then sends a controlled number of redundant packets, *i.e.* repeated packets, to fill the idle bandwidth, in order to create headroom for inaccurate arrival-time estimations caused by network jitter. We implement OLS in a Linux kernel based on MPTCP v0.95 [20] and compare its performance with the existing schedulers. Experiments show that OLS effectively reduces OFO-packets and consequently provides lower transmission latency, while maintaining a sufficient throughput in asymmetric networks with or without jitter.

The main contributions of this paper are summarized as follows.

- We propose OLS to reduce transmission latency in mobile networks with unpredictable network jitter. Basically, OLS does not seek higher throughput than needed, but uniquely trades the remaining bandwidth for lower transmission latency. To do this, OLS sends packets according to their arrival time to ensure that they arrive in order, and meanwhile sends redundant packets to avoid the impact of inaccurate arrival-time estimations when network jitter occurs.
- We analyze the performance of different scheduling algorithms to study how OFO-packets can be reduced in asymmetric networks with jitter. It illustrates that sending redundant packets effectively reduces the number of OFO-packets when network jitter occurs. And with OLS, MPTCP is more tolerant of unstable mobile networks and able to provide low transmission latency.
- OLS is implemented in a Linux kernel, with minimal cross-layer design which applications can easily get access to. And experimental evaluation shows that OLS outperforms other scheduling algorithms in reducing OFO-packets effectively while providing sufficient throughput even in a highly asymmetric network with an amount of jitter.

The rest of this paper is organized as follows. In Section II, we give a brief introduction of MPTCP and the challenges it meets, then present the motivations of our job. Related work is stated in Section III. In Section IV, the design and implementation of OLS are discussed. In Section V, we analyze the performance of different scheduling algorithms. In Section VI, we evaluate OLS with unstable heterogeneous networks. And finally, a conclusion is included in Section VII.

II. Background and Motivation

This section includes the overview of MPTCP and the challenges of it when providing low latency transmission. Then, we discuss the motivation of our proposed scheduler.

A. The Multipath TCP (MPTCP) Protocol

Mobile devices nowadays are always equipped with multiple network interfaces, allowing them to switch their access technologies when users move around. And thanks to the multipath protocols, these multi-homed devices can not only flip between several access technologies, but can also use them simultaneously to increase throughput and robustness against network failures [9], [21]. MPTCP, as a set of extensions to regular TCP, enables a transport-layer connection (a.k.a., session) to operate across multiple paths simultaneously. MPTCP operates at the transport layer and is transparent to both

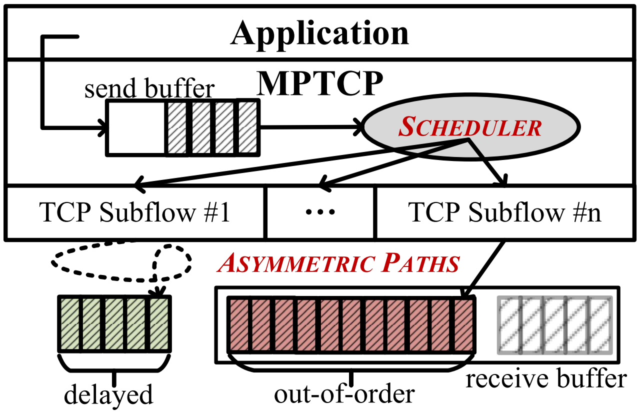


Fig. 1. An MPTCP connection over several asymmetric paths, creating OFO-packets at connection level. Data from the upper-layer application is scheduled by an MPTCP scheduler.

higher and lower layers, which means MPTCP can bring benefits to already existing applications and work well with current network middleboxes. Besides, MPTCP offers bandwidth aggregation, mobility enhancement, resilience to link failures [8], [10] and the ability to mitigate network congestion by shifting traffic away from the congested paths [22]–[26]. Since MPTCP has such attractive features, it is standardized by the IETF [4] and adopted by some major software vendors [6], [27]. Recently, MPTCP is selected to support Wi-Fi/5G co-existence in future 5G networks by 3GPP [7].

Fig. 1 illustrates how MPTCP works in the network stack. An MPTCP connection includes several subflows which act as regular TCP connections, and MPTCP scheduler takes the responsibility to split data among the subflows. For each data segment, the scheduler decides which subflow to use for transmission with respect to network properties.

B. Challenges Brought by Asymmetric and Dynamic Paths

MPTCP, just like regular TCP, provides reliable and in-order delivery, which is more challenging for a multipath protocol than a single-path one. Since MPTCP splits data among subflows, chances are that segments arrive out-of-order, especially for MPTCP operating over paths with asymmetric capacity and/or delay, which is a core use case of MPTCP [28]. When OFO-packets appear, some well-known problems may occur causing poor transmission performance [29]. For example, as shown in Fig. 1, packets on subflow No.1 are stalled due to higher delay, thus the red packets with higher sequence number become OFO-packets. These unprocessed OFO-packets are stuck in receiver's TCP buffer, causing long reordering delay, unnecessary retransmission and HoL blocking, which may greatly damage the performance of a delay-sensitive application such as live video streaming.

MPTCP scheduler plays a key role in reducing OFO-packets [29], because it is the very component that makes the decision on which packet should be sent on which subflow. The reason why current MPTCP suffers from OFO-packet problem is that in-kernel schedulers, such as Round-Robin and Lowest RTT First (LRF, the default scheduler), consider little to none about the asymmetry of multiple subflows. Some state-of-art schedulers are thus aiming at reducing OFO-packets by estimating the arrival time of packets on different subflows and make scheduling decision accordingly. But they only solve a part of the problem brought by asymmetric paths, and still remain ineffective with largely asymmetric paths or in dynamic networks where the capacity and RTT is ever changing and unpredictable. A brief survey of these schedulers is presented in Section III. Noted that MPTCP over highly asymmetric and dynamic paths is a practical scenario rather than a fictional one: as over 70 percent of the global population will have mobile connectivity by 2023 according to Cisco [1], MPTCP needs to be able to solve the OFO-problem in mobile networks with asymmetric and dynamic network properties due to heterogeneous access technologies, user movement and frequent network handoffs, etc.

It is worth mentioning that, there is one scheduler that creates few OFO-packets even under harsh network conditions, which is the ReMP scheduler [30] (a.k.a. the redundant scheduler in Linux kernel). But to this end, ReMP simply sends repeated packets on all subflows thus the benefit of aggregating bandwidth does not show up. Despite that, we actually borrow the idea of sending redundant packets to design our proposed scheduler.

C. Trading Bandwidth for Low Transmission Latency

As we mention above, there are two kinds of current schedulers, but both have drawbacks. They are:

1) Schedulers aiming at reducing OFO-packets by estimating packet arrival time precisely, which may not work well in dynamic networks.
2) Schedulers such as ReMP that are designed to achieve as low transmission latency as possible, at the cost of sacrificing the bandwidth aggregation ability of MPTCP.

In this paper, we try to solve the OFO-packet problem through taking the advantages of both. Since the mobile network conditions are unpredictable due to network jitter, to create some headroom for arrival time estimation to mitigate the effects of inaccurate estimation can be a good choice. Specifically, a scheduler can send repeated packets (*i.e.*, redundant packets) on different subflows where jitter occurs. Obviously, this scheduler concerns more about OFO-packet problem and does not seek the highest bandwidth an MPTCP connection can reach, which may be not suitable for bandwidth-hungry applications such as bulk transfer, but it is good for delay-sensitive applications with limited throughput. Typically, live streaming applications only require a certain bandwidth limited by video bitrate, and the bandwidth higher than needed (*i.e.*, remaining bandwidth) is useless. And the remaining bandwidth is exactly what we can use to send redundant packets and avoid OFO-packets.

The following section includes a brief review of some multipath transport schemes aiming at achieving low-latency transmission, and we explain how our scheduler is designed differently with other ones.

III. Related Work

Improving live streaming service quality in mobile networks has always been a research highlight [31]–[33]. And as multipath transport protocols are gradually deployed in the network [6], [7], [27], many multipath video streaming approaches are proposed in order to break the bandwidth limit of single network interface and be more resilient to link failures. For instance, MP-DASH [34] is a cross layer approach between application and transport layer, with a goal of enhancing MPTCP to support dynamic adaptive video streaming over HTTP (DASH), and ADMIT [35] is a complex MPTCP scheme with quality-driven FEC coding and rate allocation to mitigate End-to-End video streaming distortion. Song and Zhuang [36] suggested that each video burst can be dispatched to an available wireless network according to a flow splitting probability, in order to reduce the flow blocking probability and achieve a high resource utilization.

However, though lots of research has been done to validate the advantages of multipath protocols, their disadvantages are also pointed out [8]–[10]. One of the well known issues is that it is hard for multipath protocols to provide low-latency transmission services for delay-sensitive applications. And as live streaming applications emerge, the demand of low transmission latency will become ubiquitous, making it an urgent problem to be solved. Yedugundla *et al.* [29] evaluate the sufficiency of CMT-SCTP and MPTCP to transport video, web, and gaming traffic. They demonstrate that refining MPTCP scheduler is a crucial step to reduce multi-path transmission latency. In other words, though there are lots of well-designed upper-layer schemes, if MPTCP fails providing low-latency transmission at transport layer, applications are still unable to achieve expected performance. Unfortunately, MPTCP's good performance strongly relies on network properties of each subflow, and MPTCP over asymmetric paths does not outperform a single-path protocol [12], due to the occurrence of out-of-order and thus blocking. Worse still, in mobile networks, the asymmetry between WLAN and cellular can change dramatically due to network jitter.

Since it is a hard for Round-Robin and LRF to cope with asymmetric paths, several MPTCP schedulers aiming at reducing OFO-packets with asymmetric paths have been proposed. For example, DAPS (Delay-Aware Packet Scheduler [37]), which is designed as an extension to the CMT-SCTP, schedules packets according to the one-way-delay and capacities of paths in order to reduce HoL blocking over asymmetric links. But it is insensitive to dynamic network conditions change due to ignoring the queuing delay in send buffer [38], and is shown to generate spurious retransmissions [18]. To cope with this, BLEST (Blocking Estimation [18]) is designed to reducing buffer blocking. Compared with LRF, BLEST is able to skip a currently available subflow, waiting for a more advantageous subflow which can offer a lower risk of blocking. Then OTIAS (Out-of-Order Transmission for In-Order Arrival Scheduler [17]), ECF (Earliest Completion First [16]) and STTF (Shortest Transfer Time First [18]) try to estimate the arrival time of each packet more precisely by taking queuing delay in send buffer into consideration. Specifically, OTIAS builds on the idea of scheduling a segment on the subflow with the shortest transfer time, and every time a segment is to be scheduled, the OTIAS scheduler computes the expected transfer time over each of the available subflows with $(\lfloor\frac{unsent\_bytes}{cwnd}\rfloor + 0.5) \times rtt$. With similar idea, ECF uses $(\frac{unsent\_bytes}{cwnd} + 1) \times rtt$ to expect the arrival time. For more precise estimations, STTF even concerns about the congestion state of each subflow to calculate the cwnd increment in order to deal with short data flows that may finish in slow start. Besides, some other methods are used for precise estimations. For example, DPSAF (Forward Prediction based Dynamic Packet Scheduling and Adjusting with Feedback [39]) takes packet loss into consideration, and utilizes maximum likelihood estimation in TCP modeling to estimate the amount of data sent on paths simultaneously. And BCCPS (BBR-based Congestion Control and Packet Scheduling scheme) [26] designs its scheduler based on a coupled BBR algorithm and a shared bottleneck detection scheme.

Also, there are several scheduling methods designed for CMT-SCTP [40], [41] and MPQUIC [42]–[44]. To mitigate HoL blocking, the literature [40] tracks the bytes in flight and schedules packets accordingly, while another scheme [41] works in finer granularity called SCTP stream. Similarly, literarures [42]–[44] take QUIC's stream multiplexing feature into consideration and makes scheduling decisions to mitigate inter-stream blocking.

To sum up, the MPTCP schedulers mentioned above try to pursuit precise arrival-time estimations, however, it may not be a good choice especially in a network with severe and unpredictable jitter. As a result in this paper, we try to design a scheduler that creates headroom for inaccurate arrival-time estimations at the cost of remaining bandwidth, which is useless for video streaming and other applications with limited throughput. In the next section, we present our scheduler based on this distinctive idea, which turns out to be quite effective dealing with networks with jitter, and enables MPTCP provides low-latency transmission in this situation.

IV. Overlapped Scheduler for MPTCP: Design and Implementation

A. Overview

To mitigate OFO-packet problem, reduce transmission latency in mobile networks, and guarantee sufficient throughput for live video streaming, we propose OverLapped Scheduler (OLS). Generally speaking, OLS includes two scheduling algorithms, namely DElay-and-Jitter-Aware (DEJA) algorithm and ThroughPut Assurance (TPA) algorithm, which are to fulfill corresponding design objectives. The framework of OLS is shown in Fig. 2.

In order to reduce OFO-packets, DEJA is designed to send packets on the subflow with the shortest arrival time at the receiver side. DEJA estimates an arrival time for every packet on each subflow based on current send buffer size, cwnd (congestion window) and RTT of the subflow, which is a common approach in other schedulers. The uniqueness of DEJA is that it additionally takes network jitter into consideration when estimating the arrival time. And consequently

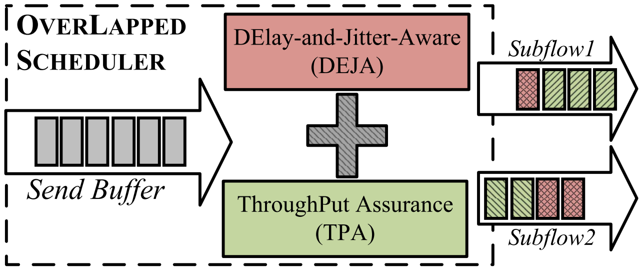


Fig. 2. The framework of OLS, including two algorithms named DEJA and TPA, which are designed to fulfill corresponding design objectives.

the arrival time of a packet on subflow i is estimated in the form of an interval $\mathbb{T}_i$, instead of a precise value. The left bound of such an interval is the arrival time estimated with the method mentioned above, and the right bound of it represents an arrival-time estimation which additionally includes the possible delay caused by network jitter. Thus when the right bound of $\mathbb{T}_i$ is smaller than the left bound of $\mathbb{T}_j$, the shortest arrival time is considered to be achieved on subflow i. However, when $\mathbb{T}_i$ and $\mathbb{T}_j$ are overlapped (*i.e.*, $\mathbb{T}_j \cap \mathbb{T}_s \neq \emptyset$), on which subflow the shortest arrival time will be achieved is stochastic. In order to make sure packets with lower sequence number arrive first, DEJA will decide to sent redundant packets, on both subflows until the two intervals are overlapped no more. By doing so, no matter there is jitter or not, the packets always arrive with successive sequence number.

On the other hand, TPA is designed to ensure sufficient throughput for upper-layer applications by limiting the number of redundant packets created by DEJA. The number of redundant packets is carefully calculated according to a given target throughput, *i.e.*, the data generation rate of upper-layer applications, which need a minor cross-layer design. TPA limits the number of redundant packets that can be sent within the current cwnd to make sure enough packets carrying new data are sent to fill the rest of the cwnd. Every time a cwnd is filled, the number of redundant packets is refreshed using the latest cwnd and RTT for another round of scheduling.

OLS functions properly on the premise that the aggregated network bandwidth is higher than the target throughput of upper-layer applications, but there is a chance that this premise is not satisfied. In this situation, OLS should spend all the bandwidth for higher throughput to meet the upper-layer requirement, thus no redundant packet is allowed to send. Though this degraded version of OLS may work unsatisfactory in networks with jitter, it can still estimate arrival time of each packet to reduce OFO-packets. Actually, in this worst case, OLS is similar to some other schedulers such as ECF and OTIAS described in Section III, and it is evaluated in Section VI as "w/o-DEJA" algorithm. Though this situation is undesirable, it will not happen for the most of the time, because applications want to avoid that situation as well, and some of them are designed to accommodate the low bandwidth connection. For example, live streaming applications using adaptive bitrate (ABR) streaming technique [45], [46] such as HTTP Live Streaming (HLS) [47] will cut down the video bitrate for smooth playback.

We present detailed descriptions about these two algorithms in the following sections. For the sake of presentation simplicity, we use RTT and smoothed RTT (srtt) interchangeably for the same meaning.

B. DElay-and-Jitter-Aware (DEJA) Scheduling Algorithm

To make packets arrive in-order at the receiver side, the basic idea behind OLS is to estimate the arrival time of every packet, and then to schedule the packet on the subflow with the shortest arrival time. Define T_i^j as the arrival time for a packet i at subflow j. Then packet i is scheduled on subflow j if T_i^j is the smallest of all subflows.

T_i^j consists of two parts, which are the time that packet i waits in the send queue of subflow j ($qtime_i^j$), and the time that packet i is inflight ($flight\_time_i^j$). In other words,

$$\begin{aligned} T_i^j &= qtime_i^j + flight\_time_i^j \\ &= \frac{qbytes_j + pkt\_size_i}{throughput_j} + \frac{srtt_j}{2}, \end{aligned} \tag{1}$$

where $qbytes_j$ and pkt_size_i denote the bytes in send queue of subflow j and the bytes of packet i, respectively. And $srtt_j$ and $throughput_j$ denotes the smoothed RTT and throughput of subflow j, respectively.

For computation and implementation efficiency, we rewrite some terms in the Eq. (1):

$$\begin{aligned} qbytes_j &= qlen_j \times MSS_j, \\ pkt\_size_i &= MSS_j, \\ throughput_j &= \frac{cwnd_j}{srtt_j} \times MSS_j, \end{aligned}$$

where $qlen_j$ and $cwnd_j$ denote the number of packets in send queue of subflow j and its congestion window size, respectively.

Since now T_i^j is a function of $srtt_j$, we might present it in the form of $T_i^j(srtt_j)$ as well. Then, we can estimate the arrival time of packet i over subflow j as follows:

$$T_i^j(srtt_j) = (\frac{qlen_j + 1}{cwnd_j} + \frac{1}{2}) \times srtt_j. \tag{2}$$

In static networks, to estimate the arrival time using $T_i^j(srtt_j)$ is feasible, but it is not the case when network jitter exists. The existence of network jitter makes it impossible to make precise estimations on the packet arrival time. As a result, the arrival time over a network with jitter can be described as an arrival-time interval $\mathbb{T}_j = (T_i^j(srtt_j), T_i^j(srtt_j + \Delta_j))$, where Δ_j characterizes the network jitter on subflow j. Δ_j can be calculated using previous RTT samples on subflow j. For example, Δ_j can be the median deviation of the RTT samples, and we use it in our in-kernel implementation. Now we can safely estimate that packet i will arrive at the receiver side within a time interval $\mathbb{T}_j$.

DEJA is presented in **Algorithm 1**. Generally, DEJA not only sends a packet on the subflow s with the shortest arrival time, but also sends the same packets to every subflow whose arrival-time interval is overlapped with that of the subflow s. Then, even though some packets may be delayed on some subflows because of network jitter, other redundant packets

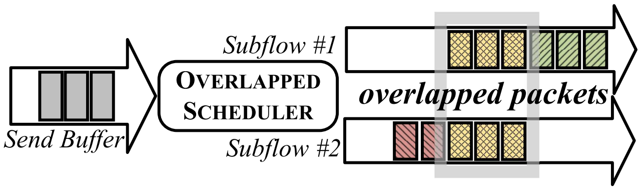


Fig. 3. OLS schedules packets from send buffer to different subflows. The yellow packets are "overlapped", which means the two subflows are transmitting repeated packets.

will fill the holes and no OFO-packet will be created. In other words, at connection level, all packets arrive in their preserved order. Packets in a send buffer will be scheduled as illustrated in Fig. 3. Redundant packets are sent on subflows with overlapped arrival-time intervals, which is where the name "overlapped scheduler" comes from.

Algorithm 1 Delay-and-Jitter-Aware Scheduling

```
Input  : packet $i$, subflow set $\mathbb{S}$
Output: selected subflows ($sel\_flows$) to send packet $i$
if $cwnds$ in all subflow are full then
    return NULL
end
for $subflow_j \in \mathbb{S}$ do
    calculate arrival time interval
    $\mathbb{T}_j = (T_i^j(srtt_j), T_i^j(srtt_j + \Delta_j))$
end
find $subflow_s$ with minimal arrival time $T_i^s$
$sel\_flows += subflow_s$
for $subflow_j \in \mathbb{S}$ and $j \neq s$ do
    if $\mathbb{T}_j \cap \mathbb{T}_s \neq \emptyset$ then
        $sel\_flows += subflow_j$
    end
end
return $sel\_flows$
```

There is one thing we need to note that in order to incur as few OFO-packets as possible, sometimes DEJA decides to send redundant packets instead of packets carrying unsent data. At worst, DEJA acts like ReMP [30] scheduler and thus may provide insufficient throughput. To tackle this issue, the throughput assurance algorithm is proposed in next section.

C. ThroughPut Assurance (TPA) Scheduling Algorithm

DEJA may trade as much bandwidth as possible for reducing the number of OFO-packets and even zero OFO-packet can be achieved when redundant packets fill the cwnd of a subflow. But it is unable to ensure sufficient throughput for upper-layer applications because of too many redundant packets. Therefore, in this section, we present TPA scheduling algorithm, which limits the number of redundant packets and guarantees good enough number of packets carrying new data.

To function properly, TPA algorithm needs a cross-layer information (see Section IV-D) which is the target throughput ($tgrt\_thp$) given by upper-layer applications, such as the video bitrate of the streaming video content. As we discuss in Section II-C, throughput higher than the target is not necessary since live streaming does not support content caching in advance. Thus, instead of seeking the maximal throughput, TPA algorithm only keeps the throughput of the MPTCP connection merely beyond the target throughput, while the remaining available bandwidth can be used by DEJA to send redundant packets for low transmission latency.

To achieve the goal, TPA is designed to limit the number of redundant packets. A parameter red_thp_j is introduced to represent the total throughput that can be used to sent redundant packets by subflow j. TPA calculates red_thp_j as

$$\begin{aligned} red\_thp_j &= throughput_j - rsv\_thp_j \\ &= \frac{cwnd_j \times MSS_j}{srtt_j} - rsv\_thp_j, \end{aligned} \tag{3}$$

where rsv_thp_j is the reserved throughput which is needed to send packets carrying new data and ensure sufficient throughput. And the reserved throughput of subflow j is

$$rsv\_thp_j = tgrt\_thp - \sum_{i \in \mathbb{S}, i \neq j} \frac{cwnd_i \times MSS_i}{srtt_i},$$

and if $rsv\_thp_j < 0$, set it to zero. Note that $rsv\_thp_j > throughput_j$ may happen when the MPTCP connection does not have sufficient bandwidth to carry the traffic with such a high target throughput. In this situation, red_thp is set to zero and no redundant packet is allowed to send. It is also worth mentioning that TPA estimates $throughput_j$ using MSS_j instead of pkt_size_j in Eq. (2). In this way, only the payload of a packet (excluding the header) is used to calculate the throughput, and thus $throughput_j$ represents the actual throughput that an application can get. In other words, $throughput_j$ matches the given target throughput at application level, and no conversion is needed when live streaming applications set their video bitrate as the target throughput.

TPA works as follows. When the first packet is scheduled on subflow j, TPA initiates red_thp_j according to Eq. (3). Then whenever a packet is scheduled by DEJA on subflow j, TPA intervenes the scheduling decision according to **Algorithm 2**. Specifically, TPA decreases red_thp_j, and when $red\_thp_j \leq 0$, the redundant packet is not allowed to send. Instead, TPA waits until packets carrying fresh data are scheduled by DEJA to fill the remaining $cwnd_j$. Finally when current $cwnd_j$ is filled, red_thp_j is refreshed for the next round of scheduling.

By combining DEJA and TPA algorithms, we finally get OLS. OLS is a new way to balance the trade-off between reduced number of OFO-packets and high throughput. Specifically, some existing schedulers, such as LRF, try to use all available bandwidth to transmit fresh data for maximal throughput, but consequently create lots of OFO-packets. In contrast, schedulers like ReMP send repeated data through all the subflows, creating no OFO-packets but providing low throughput. OLS is an elastic design that guarantees

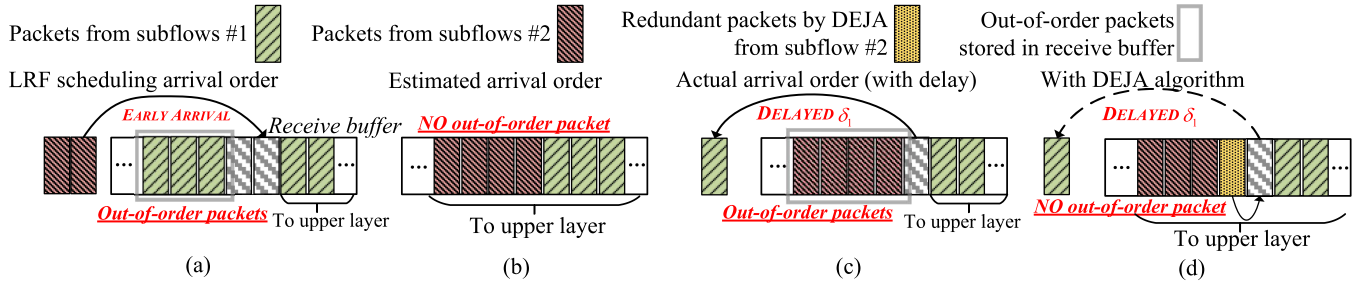


Fig. 4. Packets arrive in receive buffer with different scheduling algorithms, creating OFO-packets due to asymmetric paths with network jitter. Packets with high sequence number are on the left side of the receive buffer.

Algorithm 2 TPA's Actions on Packet Scheduling

```
if a redundant packet is scheduled on $subflow_j$ then
    if $red\_thp - \frac{MSS_j}{srtt_j} > 0$ then
        $red\_thp -= \frac{MSS_j}{srtt_j}$
        redundant packet is allowed to send
    else
        discard the scheduling decision
    end
end
if $cwnd_j$ is filled then
    recalculate $red\_thp$ with Eq. (3)
end
```

sufficient, though not maximal, throughput and at the same time also keeps the number of OFO-packets as small as possible.

D. Implementation of the Cross-Layer Design in TPA Algorithm

In this subsection, we explain how OLS is implemented in a Linux kernel and how an application react with OLS through easy access. In short, we use the *proc file system (procfs)* to achieve information interaction between kernel modules and user processes [48]. The *procfs* provides some interfaces to the kernel variables in the form of some writable files, which is mounted at */proc* like common files in Linux kernel by default. By modifying these files, some parameters in Linux kernel modules can be changed.

In the case of OLS, since it is implemented as a kernel module, it can use *proc_create* function to create a writable pseudo-file which contains a variable that can be modified by applications. All an application needs to do is to write its target throughput to the pseudo-file. Then OLS can read from the file and will automatically keep the actual throughput above it with our TPA algorithm.

Noted that helping an application setting their target bandwidth is out of scope for our job, and some applications are aware of their target throughput naively. For instance, streaming servers are aware of the bitrate of their video content according to its resolution, frame rate, encoder, etc. And these servers allow their users to select one of the several video qualities given, manually or with mechanisms such as ABR [49]. Thus when an MPTCP connection is established for video streaming, the target throughput can be directly set as the video bitrate. Since TPA algorithm only concerns about the payload of a packet and ignoring the header overhead, its estimated throughput matches the given target throughput at application level and need no conversion.

V. Discussion on DEJA Scheduling Algorithm

In this section, we analyze the behaviors of three scheduling algorithms about generating OFO-packets in networks with jitter, which are LRF as a baseline, "w/o-DEJA" (defined in Section VI) scheduling algorithm that sends no redundant packet, and the proposed DEJA scheduling algorithm.

We focus on one of the most common scenarios of an MPTCP connection with two subflows, namely subflow 1 and subflow 2. For the convenience of explanation, we assume that RTT of subflow 2 is larger than that of subflow 1 ($srtt_2 > srtt_1$). In following discussions, we calculate the theoretical maximum OFO-packets. Fig. 4 presents the situation of receive buffer with different scheduling algorithms and network conditions.

A. Lowest RTT First (LRF)

The number of OFO-packets created by LRF scheduler can be simply calculated as:

$$ofo\_size = (\frac{srtt_2}{2 \times srtt_1} - 1) \times cwnd_1 + incre\_cwnd_1,$$

where $incre\_cwnd_1$ is the increment of $cwnd_1$ in $\frac{srtt_2}{2}$. It can be calculated as:

$$incre\_cwnd_1 = (\frac{dRTT - 1}{2}) \times dRTT,$$

$$dRTT = \frac{srtt_2}{srtt_1}.$$

As illustration in Fig. 4(a), OFO-packets are created because LRF schedules packets on subflow 2 when $cwnd_1$ is full. However, packets sent from subflow 1 in subsequent cwnds may arrive earlier. They have higher sequence number and form queue in receive buffer. The queue becomes longer with greater RTT difference and bigger $cwnd_1$. Thus, LRF cannot handle an MPTCP connection with asymmetric paths. Note that theoretically, all OFO-packets will be passed to upper layer within $\frac{srtt_2}{2}$.

B. W/o-DEJA

All packets are scheduled according to their arrival time by w/o-DEJA, thus they arrive at the receiver side in order. These in-order packets can be immediately passed to upper-layer application, creating no queue as shown in Fig. 4(b).

But when network jitter occurs, for example a packet on subflow 1 is delayed for δ_1 which is unexpected by the scheduler, the packets with higher sequence will queue up in buffer. Then the maximal OFO queue will be generated right before the delayed packet arrives as shown in Fig. 4(c). Then we can calculate the number of OFO-packets as follows:

$$\begin{aligned} ofo\_size &= min\{\frac{throughput_2}{MSS_2} \times \delta_1, cwnd_2\} \\ &= min\{\frac{cwnd_2}{srtt_2} \times \delta_1, cwnd_2\}. \end{aligned}$$

Compared with LRF, asymmetric RTTs are not the cause of OFO-packets for w/o-DEJA, but network jitter remains a problem according to the δ_1 term in the equation.

C. DEJA

DEJA defines arrival time as an interval $\mathbb{T}_j$ instead of a precise value T_i^j like w/o-DEJA does. This interval creates some headroom for possible errors due to jitter, and finally reduce OFO-packets. In static networks, DEJA theoretically creates no OFO-packet, either. Then, an unexpected delay occurs at a packet on subflow 1 for δ_1 as shown in Fig. 4(d). Different from w/o-DEJA, DEJA is aware of the jitter thus schedules some redundant packets. These packets are the keys to offset the effect of jitter, so we estimate the number of them.

According to **Algorithm 1**, DEJA schedules identical packets on both subflows when $\mathbb{T}_1 \cap \mathbb{T}_2 \neq \emptyset$. In our scenario, subflow 2 is relatively stable, thus $\mathbb{T}_2 = T^2(srtt_2)$ (ignoring the packet number here for expression convenience). At the beginning, we have $T^2(srtt_2) > T^1(srtt_1 + \Delta_1)$ and packets are scheduled on subflow 1. The moment subflow 2 begins to send redundant packets when $T^2(srtt_2) = T^1(srtt_1 + \Delta_1)$ and after sending N redundant packets, subflow 2 stops sending another redundant packets under two different situations. Here we denote the number of redundant packets is N and after N scheduled packets, the estimated arrival time of subflow i is denoted as T_{N+1}^i. The subflow 2 stops sending another redundant packets when:

1) $T_{N+1}^2(srtt_2) > T_{N+1}^1(srtt_1 + \Delta_1)$, which means subflow 2 only sends redundant packets and thus it creates no OFO-packet. That may be caused by the mitigated jitter on subflow 1 or $T_{N+1}^2(srtt_2)$ increases rapidly because of a big $srtt_2$. Though subflow 2 may creates OFO-packets later, the number of them will not outnumber the theoretical maximum in next situation.
2) $T_{N+1}^2(srtt_2) < T_{N+1}^1(srtt_1)$, which means the estimated arrival time on subflow 2 is shorter than that on subflow 1. From this moment on, DEJA schedules new packets on subflow 2 to fill the space left in $cwnd_2$, and the number is $cwnd_2 - N$. To illustrate how OFO-packets are created by DEJA, we need to calculate the number of redundant packets N first, we have

$$\begin{aligned} N &= throughput_1 \times (T_{N+1}^1(srtt_1) - T^1(srtt_1)) \\ &= throughput_1 \times (T^1(srtt_1 + \Delta_1) - T^1(srtt_1)) \\ &= \frac{cwnd_1}{srtt_1} \times (\frac{qlen_j + 1}{cwnd_j} + \frac{1}{2}) \times \Delta_1. \end{aligned}$$

With the constraint of $cwnd_2$, we finally get:

$$N = min\{\frac{cwnd_1}{srtt_1} \times (\frac{qlen_j + 1}{cwnd_j} + \frac{1}{2}) \times \Delta_1, cwnd_2\}. \quad (4)$$

If the estimation of jitter Δ_1 covers the actual packet delay δ_1 on subflow 1, no OFO-packet is created. Because in this situation, even though packets (less than N) on subflow 1 is delayed, their copies arrive at receiver from other subflows in order.

DEJA generates OFO-packets only when the actual delay of the packets on subflow 1 is beyond our estimation, and no redundant packet can fill the hole. In this situation, DEJA creates the same number of OFO-packets as w/o-DEJA does.

In a word, DEJA only creates OFO-packets when network jitter is severe, more specifically, when the unexpected delayed packets number N_u exceeds N in Eq. (4). That is,

$$ofo\_size = min\{\frac{cwnd_2}{srtt_2} \times \delta_1, cwnd_2\},$$

which will soon disappear after the $N_u - N$ delayed packets arrives.

VI. Performance Evaluation

A. Experimental Setup

Performance evaluation is presented in this section. For performance comparison, several well known schedulers, such as Round-Robin, LRF, and BLEST, are also evaluated in our evaluation. Besides, in addition to the full-featured OLS, we also disable DEJA or TPA algorithm separately to test their functions under different circumstances. We denote OLS disabling DEJA and TPA as w/o-DEJA and w/o-TPA respectively. Descriptions about them are as follows:

- W/o-DEJA is an important algorithm as a contrast to OLS. It can be regarded as a degraded version of OLS that ignores network jitter ($\Delta_j = 0, \forall j$), which means it sends no redundant packets. Its performances correspond to existing MPTCP schedulers that try to reduce OFO-packets with arrival-time estimations, such as ECF [16] and OTIAS [17] (described in Section III). Furthermore, it can also be regarded as a OLS when the video generation rate is higher than the total bandwidth ($red\_thp = 0$, for all subflows). We implement w/o-DEJA by simply setting $\Delta_j = 0$ in **Algorithm 1**, or setting $red\_thp = 0$ in **Algorithm 2**.
- W/o-TPA does not limit the number of redundant packets that DEJA creates, thus it does not ensure sufficient throughput ($red\_thp = cwnd$, for all subflows). As a comparison, its performances illustrate that OFO-packets can be greatly reduced by sacrificing throughput, and

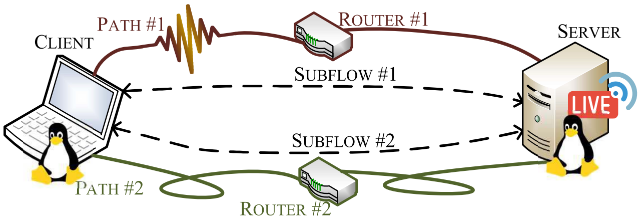


Fig. 5. The network topology for performance evaluation.

shows how OLS balances the trade-off between reduced number of OFO-packets and high throughput. We implement w/o-TPA by simply setting $red\_thp - \frac{MSS_j}{srtt_j} > 0$ in **Algorithm 2**.

We conduct two experiments in our testbed. As shown in Fig. 5, our testbed has a common topology in real mobile networks, e.g. an MPTCP connection with two subflows, using 4G and Wi-Fi respectively. Though the topologies of the experiments are the same, different network conditions of the two subflows are introduced in the two routers in Fig. 5 with *tc* in Linux kernel, such as RTTs, capabilities and the existence of network jitter. To simulate the performance of a live video streaming server, the server provides a data stream of a certain rate for the client with *iperf3*, which is also the target throughput of TPA. The client and the server are MPTCP-enabled with OLS, based on MPTCP v0.95 [20].

B. OFO-Packet Reduction in Static Asymmetric Network

The first experience illustrates the superiority of OLS in reducing OFO-packets in static asymmetric network. In this experiment, the server offers a 10 Mbps traffic through the two paths of different RTTs to mimic an application generating 10 Mbps video streaming traffic. The bandwidth of the two paths is 6 Mbps respectively, which means one path provides insufficient throughput while two paths provides more than necessity. The subflow 1 runs on a 20 ms-RTT path and the RTTs of path 2 are set from 20 ms to 160 ms in order to fully investigate the performance of OLS in different asymmetric scenarios. Our settings borrow some ideas from Hurtig *et al.* [18], who fix the RTT of one path and the RTT of another path covers a wide range.

Compared with other in-kernel schedulers, such as BLEST, LRF and Round-Robin (RR), OLS can reduce OFO-packets thanks to the ***basic idea*** of estimating packets arrival time. Up to 80% of the OFO-packets are eliminated using OLS in very asymmetric networks. Note that the actual throughput of OLS is almost equal to that of the in-kernel schedulers (Table I), which means the same number of packets are sent to the client every moment. However, not all of them are handed immediately to upper-layer applications. Instead, they stay unprocessed in OFO queue and have to wait for the packets with lower sequence number, which are supposed to arrive earlier. As shown in Fig. 6, OLS generates much less OFO queue per 0.1 second, which shows its superiority on providing low-latency transmission.

We also evaluate w/o-DEJA and w/o-TPA. In static networks, they also successfully reduce the number of OFO-packets. However, we can observe from Fig. 6 that w/o-DEJA creates a little more OFO-packets than full-featured OLS. In addition, even though w/o-TPA creates the least OFO-packets, it only provides a throughput less than 8.48 Mbps as shown in Table I, while other schedulers reach at least 9.57 Mbps. These disadvantages are even amplified by network jitter as we present in next subsection.

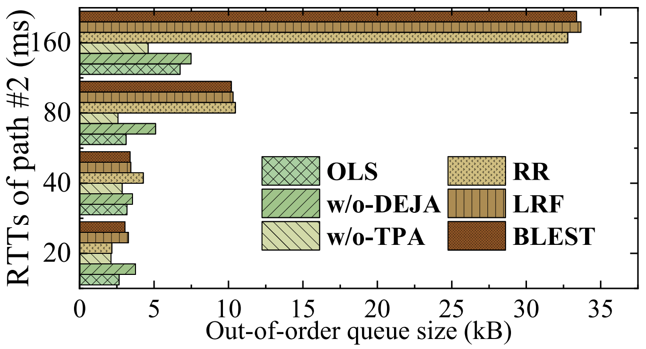


Fig. 6. The average size of OFO queue that is created every 0.1 second in static asymmetric networks by different schedulers. The RTT of subflow 1 is 20 ms.

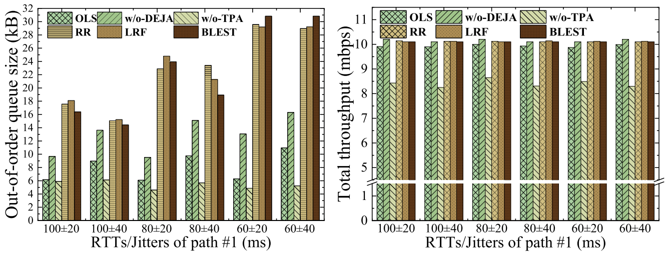
TABLE I
THROUGHPUT OF SCHEDULERS IN STATIC ASYMMETRIC NETWORKS

Schedulers	**RTTs of Path 2**			
	20 ms	40 ms	80 ms	160 ms
BLEST	9.96 Mbps	9.97 Mbps	9.95 Mbps	9.61 Mbps
LRF	9.98 Mbps	9.96 Mbps	9.87 Mbps	9.85 Mbps
RR	9.98 Mbps	9.95 Mbps	9.95 Mbps	9.61 Mbps
OLS	10.1 Mbps	9.95 Mbps	9.87 Mbps	9.57 Mbps
w/o-DEJA	10.0 Mbps	9.97 Mbps	9.96 Mbps	9.88 Mbps
w/o-TPA	8.48 Mbps	8.29 Mbps	7.72 Mbps	7.21 Mbps

C. OFO-Packet Reduction in a Network With Jitter

Mobile networks are full of jitter caused by issues such as users movement, limited range of wireless signals, etc. For example, Li *et al.* [10] illustrated that significant RTT spikes appear before a network handoff, and thus the OFO queue size rises with the jitter and increase in the RTT of subflows, especially around moments of handoffs. Furthermore, they also indicated that since only in-order packets can be used by upper-layer applications, the total throughput may be higher than the actual data rate due to OFO-packet problem.

As a result, in the second experiment, RTT jitter is introduced to one of the two subflows to simulate the situation before a handoff in real mobile networks. We treat the severity of the RTT jitter and increase as the variable in this experiment, and evaluate the efficiency of different schedulers in reducing OFO-packets. Though OLS, w/o-DEJA and w/o-TPA work well in static networks, network jitter may magnify the OFO-packet problem and may make it harder to handle. Note that though implemented with some difference, w/o-DEJA is actually the idea that most of the schedulers use to reduce OFO-packets without caring about network jitter. So by testing w/o-DEJA, we can find out if OLS has a superiority over existing schedulers in reducing OFO-packets and transmission latency in networks with jitter.



(a)

(b)

Fig. 7. The performances of different schedulers in networks with jitter. (a) The average size of OFO-packets they create per 0.1 second; (b) The total throughput they provide for the upper-layer application.

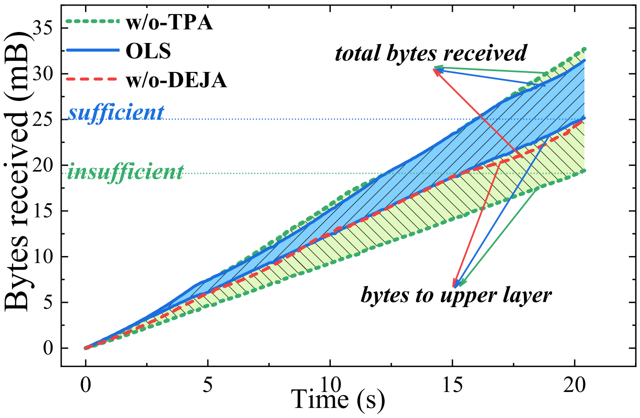


Fig. 8. The total bytes received from the both subflows and the actual number of bytes that are passed to the upper-layer application. The RTT of path 1 is 80±40 ms. The shaded areas represent extra redundant data, and the two lines of w/o-DEJA are overlapping.

In this experiment, the server provides a 10 Mbps traffic over the two paths with the same bandwidth of 8 Mbps, to mimic a streaming application with 10 Mbps bitrate. The RTT of path 2 is 160 ms, and the RTT of path 1 is with jitter. In the following description and figures, expressions like "80±40 ms" denote that the RTT of path 1 is changing randomly from 40 ms to 120 ms. Intuitively, schedulers may prefer path 1 with lower average RTT (e.g., LRF) and send more packets through it, as a result, when this path suffers jitter and the arrival-time estimations are inaccurate, more OFO-packets would be created.

We record the average OFO queue size, the total throughput at connection level and bytes received at each subflow in a period of twenty seconds during transmission. The statistics are presented in Fig. 7.

First, OLS, w/o-DEJA and w/o-TPA create far fewer OFO-packets than some in-kernel schedulers, namely Round-Robin and LRF. As shown in 7a, though network jitter has effects on these schedulers, creating more OFO-packets when RTT jitter is severer, the major factor is actually the difference of average RTTs, *i.e.*, asymmetry of RTTs, between two paths. Surprisingly, BLEST fails to reduce OFO-packets in both experiments. Apparently, to use existing in-kernel schedulers in asymmetric networks is not a wise choice.

Though w/o-DEJA creates fewer OFO-packets than in-kernel schedulers, we can observe from Fig. 7a that w/o-DEJA creates more OFO-packets twice as many as what OLS does (13.1 kB to 6.3 kB, at the $60 \pm 20\ ms$ column). The reason is that w/o-DEJA ignores network jitter. As a result, when estimating the arrival time of certain packets, w/o-DEJA makes lots of mistakes about on which subflow the packets will arrive earlier because of the jittering RTT of path 1. Another thing that we observe from the figure is that the median of the RTT of path 1 only has minor effect on OFO-packets, and the range of RTT jitter is the most influential factor, which confirms our theory in Section V.

Except for OFO-packets, sufficient throughput is also what qualified schedulers need to ensure and w/o-TPA is apparently not one of them. We mentioned in Section IV that without TPA, unlimited redundant packets are sent to fill the cwnd and at worse it works like ReMP. The throughput of w/o-TPA in Fig. 7b is slightly higher than 8 Mbps – the bandwidth of a single path, which is much lower than the target rate 10 Mbps. And the other schedulers manage to maintain a throughput around 10 Mbps. Note that the throughput of OLS is slightly lower than the target because of some implementation issues such as inaccurate throughput estimation with cwnd and RTT. Since the prototype of OLS we implemented is functionally good enough, we leave the work of refinement for the future.

We take a closer look at how OLS works when the RTT of path 1 is 80±40 ms. In Fig. 8, we present the bytes received by the client. In the context of MPTCP, the total number of bytes received from both subflows are equal to or larger than that are finally passed to upper layer (marked respectively with text in Fig. 8), because redundant packets will be dropped by MPTCP. For w/o-TPA and OLS, the number of bytes received from the two subflows are more than what upper-layer applications actually need, because redundant packets are sent to reduce OFO-packets. On the other hand, w/o-DEJA and OLS pass the same number of bytes (25.0 MB and 25.2 MB respectively) to upper layer within twenty seconds, but while providing the same throughput, OLS utilizes more bandwidth. The total bytes sent to the two subflows by OLS is 31MB, and the 6 MB redundant bytes are the key of fewer OFO-packets. Note that the server provides traffic with a limited rate just like real live streaming server, thus it is impossible to reach a throughput higher than 10 Mbps. OLS actually makes full use of the available bandwidth aggregated by MPTCP, not for higher throughput, but for fewer OFO-packets, in other words, for lower latency. While o/w-DEJA does not utilize available bandwidth, it is indeed a waste of the aggregated bandwidth.

Now, we have a direct observation of the reordering delay that OFO-packets wait for in OFO queue in Fig. 9. The in-kernel schedulers not only create numerous OFO-packets, but also have more than 10% of them stuck in receive buffer for more than 60 ms (BLEST) or 125 ms (Round-Robin and LRF). In a word, compared with other three schedulers, the in-kernel schedulers many more OFO-packets stuck in receive buffer for a longer period. On the other hand, compared with OLS and w/o-TPA, w/o-DEJA generates more OFO-packets, and also makes them wait for longer periods in queue before they are passed to upper layer. For example, more than 10% of the OFO-packets wait for longer than 125 ms in queue,

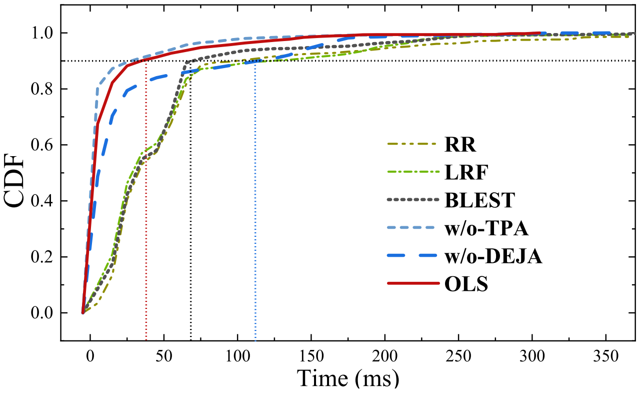


Fig. 9. The time that packets wait and stay unprocessed in OFO queue. The RTT of path 1 is 80±40 ms.

which is almost equally poor to some in-kernel schedulers. Since the RTT of path 2 is only 160 ms, 125 ms has severe influence on transmission latency. On the contrary, over 90% of the OFO-packets are freed within 35 ms by OLS, which is a much shorter period. This phenomenon confirms the validity of our algorithm that even though some packets may be stalled on one subflow because of jitter, the same packets arrive soon through another subflow and empty the OFO queue. This figure provides a clear illustration that OLS indeed reduces the reordering latency, showing its superiority over other schedulers to provide low-latency transmission in networks with jitter.

In a word, OLS reduces the number of OFO-packets, provides low latency transmission, and ensures sufficient throughput, even in networks with a considerable amount of jitter like mobile networks. The comparison between OLS and w/o-DEJA illustrates that our core idea of trading bandwidth for low transmission latency is efficient, and TPA also shows its importance in throughput assurance.

VII. Conclusion

Multipath TCP, as an IETF standard supposed to outperform single-path TCP, fails providing low-latency transmission in mobile networks, where asymmetric network paths and network jitter are common. The reason for that failure can be found in the scheduler which decides over which interface to transmit data. In this paper, we proposed OLS in order to provide low-latency transmission and sufficient throughput for live video streaming applications and other delay-sensitive applications consuming limited bandwidth by tackle the OFO-packet problem. To reduce OFO-packets, OLS sends packets according to the estimated arrival time of each packet, and instead of pursuing precise estimation, OLS sends a controlled number of redundant packets on certain subflows when network jitter occurs so as to make sure in-order arrivals at MPTCP connection level. We implement OLS in a Linux kernel and experimental evaluation shows that it outperforms existing schedulers in creating fewer number of OFO-packets while maintaining a target throughput even in the networks with a considerable amount of jitter. In a word, MPTCP with OLS is fully capable to meet the requirements of live streaming applications as well as other delay-sensitive applications.

References

[1] (2020). *Cisco Annual Internet Report (2018-2023) White Paper*. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2] 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.

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

[4] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, and C. Paasch, *TCP Extensions for Multipath Operation With Multiple Addresses*, document RFC 6824, IETF, 2020, Accessed: May 2021. [Online]. Available: https://www.ietf.org/rfc/rfc8684.txt

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

[6] *Advances in Networking, Part 1 and Part 2, 2017, in the 2017 Apple Worldwide Developers Conference*. Accessed: May 2021. [Online]. Available: https://developer.apple.com/videos/wwdc2017/

[7] *System Architecture for the 5G System (5GS); Stage 2 (Release 16)*, document 3GPP TS 23.501, V16.7.0, 2020. Accessed: Nov. 2020. [Online]. Available: https://www.3gpp.org/ftp//Specs/archive/23_series/23.501/23501-g70.zip

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

[9] 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 (SIGCOMM)*, 2011, pp. 266–277.

[10] 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.*, Aug. 2018, pp. 161–175.

[11] K. Xue, J. Han, H. Zhang, K. Chen, and P. Hong, "Migrating unfairness among subflows in MPTCP with network coding for wired–wireless networks," *IEEE Trans. Veh. Technol.*, vol. 66, no. 1, pp. 798–809, Jan. 2017.

[12] S. Ferlin, T. Dreibholz, and O. Alay, "Multi-path transport over heterogeneous wireless networks: Does it really pay off?" in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2014, pp. 4807–4813.

[13] A. Ramachandran and D. M. Kantarcioglu, "Using blockchain and smart contracts for secure data provenance management," 2017, *arXiv:1709.10000*. [Online]. Available: http://arxiv.org/abs/1709.10000

[14] M. Polese, F. Chiariotti, E. Bonetto, F. Rigotto, A. Zanella, and M. Zorzi, "A survey on recent advances in transport layer protocols," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3584–3608, Aug. 2019.

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

[16] Y.-S. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "ECF: An MPTCP path scheduler to manage heterogeneous paths," in *Proc. Int. Conf. Meas. Model. Comput. Syst. (ACM SIGMETRICS)*, Jun. 2017, pp. 147–159.

[17] 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 (AINAW)*, May 2014, pp. 749–752.

[18] 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.

[19] 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.

[20] *Multipath TCP in the Linux Kernel*. Accessed: May 2021. [Online]. Available: http://www.multipath-tcp.org/

[21] M. Kheirkhah and M. Lee, "A solution to MPTCP's inefficiencies under the incast problem for data center networks," *Comput. Commun.*, vol. 161, pp. 238–247, Sep. 2020.

[22] W. Guo *et al.*, "Delay-based congestion control for multipath TCP," in *Proc. Adv. Multimedia, Commun. Netw. (ICNP)*, Dec. 2013, pp. 1–10.

[23] C. Raiciu, M. Handley, and D. Wischik, *Coupled Congestion Control for Multipath Transport Protocols*, document RFC 6356, IETF, 2011. Accessed: May 2021. [Online]. Available: https://www.ietf.org/rfc/rfc6356.txt

[24] R. Khalili, N. Gast, M. Popovic, and J.-Y. L. 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.

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

[26] 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.

[27] O. Bonaventure. (2019). *Apple Music on IOS13 Uses Multipath TCP Through Load-Balancers*. Accessed: May 2021. [Online]. Available: http://blog.multipath-tcp.org/blog/html/2019/10/27/apple_music_on_ios13_uses_multipath_tcp_through_load_balancers.html

[28] O. Bonaventure, C. Paasch, and G. Detal, *Use Cases and Operational Experience With Multipath TCP*, document RFC 8041, IETF, 2017. Accessed: May 2021. [Online]. Available: https://www.ietf.org/rfc/rfc8041.txt

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

[30] A. Frommgen, T. Erbshäußer, 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.

[31] D. Huang, X. Tao, C. Jiang, S. Cui, and J. Lu, "Trace-driven QoE-aware proactive caching for mobile video streaming in metropolis," *IEEE Trans. Wireless Commun.*, vol. 19, no. 1, pp. 62–76, Jan. 2020.

[32] D. Bethanabhotla, G. Caire, and M. J. Neely, "WiFlix: Adaptive video streaming in massive MU-MIMO wireless networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 6, pp. 4088–4103, Jun. 2016.

[33] R. Atawia, H. S. Hassanein, and A. Noureldin, "Robust long-term predictive adaptive video streaming under wireless network uncertainties," *IEEE Trans. Wireless Commun.*, vol. 17, no. 2, pp. 1374–1388, Feb. 2018.

[34] B. Han, F. Qian, L. Ji, and V. Gopalakrishnan, "MP-DASH: Adaptive video streaming over preference-aware multipath," in *Proc. 12th Int. Conf. Emerg. Netw. EXperiments Technol. (CoNEXT)*, Dec. 2016, pp. 129–143.

[35] J. Wu, C. Yuen, B. Cheng, M. Wang, and J. Chen, "Streaming high-quality mobile video with multipath TCP in heterogeneous wireless networks," *IEEE Trans. Mobile Comput.*, vol. 15, no. 9, pp. 2345–2361, Sep. 2016.

[36] W. Song and W. Zhuang, "Performance analysis of probabilistic multipath transmission of video streaming traffic over multi-radio wireless devices," *IEEE Trans. Wireless Commun.*, vol. 11, no. 4, pp. 1554–1564, Apr. 2012.

[37] 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 (AINAW)*, Mar. 2013, pp. 1119–1124.

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

[39] 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.

[40] I. A. Halepoto, F. C. M. Lau, and Z. Niu, "Scheduling over dissimilar paths using CMT-SCTP," in *Proc. 7th Int. Conf. Ubiquitous Future Netw.*, Jul. 2015, pp. 535–540.

[41] J. Eklund, K.-J. Grinnemo, and A. Brunstrom, "Efficient scheduling to reduce latency for signaling traffic using CMT-SCTP," in *Proc. IEEE 27th Annu. Int. Symp. Pers., Indoor, Mobile Radio Commun. (PIMRC)*, Sep. 2016, pp. 1–6.

[42] J. Wang, Y. Gao, and C. Xu, "A multipath QUIC scheduler for mobile HTTP/2," in *Proc. 3rd Asia–Pacific Workshop Netw. (APNet)*, Aug. 2019, pp. 43–49.

[43] X. Shi, L. Wang, F. Zhang, B. Zhou, and Z. Liu, "PStream: Priority-based stream scheduling for heterogeneous paths in multipath-QUIC," in *Proc. 29th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2020, pp. 1–8.

[44] A. Rabitsch, P. Hurtig, and A. Brunstrom, "A stream-aware multipath QUIC scheduler for heterogeneous paths," in *Proc. Workshop Evol., Perform., Interoperability (QUIC)*, Dec. 2018, pp. 29–35.

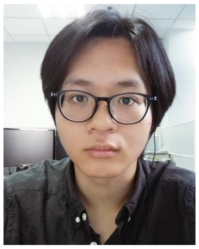
[45] S. Akhshabi, A. C. Begen, and C. Dovrolis, "An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP," in *Proc. 2nd Annu. ACM Conf. Multimedia Syst.*, Feb. 2011, pp. 157–168.

[46] O. Oyman and S. Singh, "Quality of experience for HTTP adaptive streaming services," *IEEE Commun. Mag.*, vol. 50, no. 4, pp. 20–27, Apr. 2012.

[47] R. Pantos and W. May, *HTTP Live Streaming*, document RFC 8216, IETF, 2017. Accessed: May 2021. [Online]. Available: https://www.ietf.org/rfc/rfc8216.txt

[48] M. Kerrisk. (2021). *Proc(5)—Linux Manual Page*. Accessed: May 2021. [Online]. Available: https://man7.org/linux/man-pages/man5/proc.5.html

[49] Y. Sani, A. Mauthe, and C. Edwards, "Adaptive bitrate selection: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2985–3014, Jul. 2017.

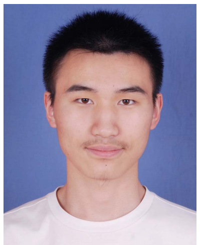


Yitao Xing received the B.S. degree in information security from the School of the Gifted Young, University of Science and Technology of China (USTC), in 2018. He is currently pursuing the Ph.D. degree in information security with the School of Cyber Security. His research interests include future Internet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

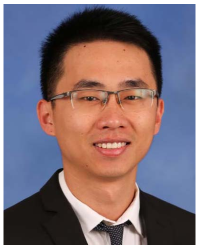
Kaiping Xue (Senior Member, IEEE) received the bachelor's degree from the Department of Information Security, University of Science and Technology of China (USTC), in 2003, and the Ph.D. degree from the Department of Electronic Engineering and Information Science (EEIS), USTC, in 2007. From May 2012 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 Security and the Department of EEIS, USTC. His research interests include future Internet architecture, transmission optimization, distributed networks, and network security. He is a fellow of the IET. He serves on the Editorial Board for several journals, including the IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS (TWC) and the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT (TNSM). He has also served as a (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 B.S. degree from the Department of Electronic Engineering and Information Science (EEIS), USTC, in July 2019, where he is currently pursuing the Graduate degree in communication and information system. His research interests include future Internet architecture design and transmission optimization.

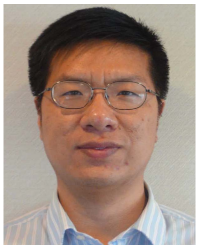
Jiangping Han (Graduate Student Member, IEEE) received the B.S. degree from the Department of Electronic Engineering and Information Science (EEIS), USTC, in July 2016, where she is currently pursuing the Ph.D. degree in communication and information systems. Her research interests include future Internet architecture design and transmission optimization.



Jian Li (Member, IEEE) received the B.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. He is currently a Post-Doctoral Researcher with the School of Cyber Security, USTC. His research interests include wireless communications, satellite networks, and future Internet architecture.



Jianqing Liu (Member, IEEE) received the B.Eng. degree from the University of Electronic Science and Technology of China, Chengdu, China, in 2013, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA, in 2018. He is currently an Assistant Professor with the Department of Electrical and Computer Engineering, The University of Alabama in Huntsville. His research interests include wireless networking and network security in cyber-physical systems.



Ruidong Li (Senior Member, IEEE) received the B.Eng. degree from Zhejiang University, China, in 2001, and the D.Eng. degree from the University of Tsukuba in 2008. He is currently an Associate Professor with the College of Science and Engineering, Kanazawa University, Japan. Before joining Kanazawa University, he was a Senior Researcher with the Network System Research Institute, National Institute of Information and Communications Technology (NICT). His current research interests include future networks, big data networking, blockchain, information-centric networks, the Internet of Things, network security, wireless networks, and quantum Internet. He is a member of IEICE. He is the Founder and the Chair of the IEEE SIG on big data intelligent networking and IEEE SIG on intelligent Internet edge, and the Secretary of IEEE Internet Technical Committee. He also serves as the Chair for conferences and workshops, such as IWQoS 2021, MSN 2020, BRAINS 2020, ICC 2021 NMIC Symposium, ICCN 2019/2020, and NMIC 2019/2020; and organized the special issues for the leading magazines and journals, such as *IEEE Communications Magazine*, IEEE NETWORK, and IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING (TNSE).