A Stream-Aware MPQUIC Scheduler for HTTP Traffic in Mobile Networks

Yitao Xing, *Graduate Student Member, IEEE*, Kaiping Xue, *Senior Member, IEEE*, Yuan Zhang, Jiangping Han, *Member, IEEE*, Jian Li, *Member, IEEE*, David S. L. Wei, *Life Senior Member, IEEE*, Ruidong Li, *Senior Member, IEEE*, Qibin Sun, *Fellow, IEEE*, and Jun Lu

***Abstract*— A QUIC (Quick UDP Internet Connections) protocol is designed to improve Hypertext Transfer Protocol (HTTP) traffic and carries a non-negligible portion of the traffic in the current Internet. As its extension, Multipath QUIC (MPQUIC) provides higher bandwidth and smoother network handover by using multiple network interfaces simultaneously. However, to improve HTTP traffic, there are still some issues not yet carefully addressed in the existing MPQUIC, and packet scheduling is a vital one among the issues. Specifically, existing methods fail to respond to the stream prioritization of HTTP Version 2 (HTTP/2), leading to unsatisfying web page load performance. Besides, managing asymmetric and dynamic network paths is also a challenging issue, which may result in Head-of-Line (HoL) blocking and excessive buffer usage if not effectively handled. In this paper, we present a stream-aware per-packet scheduler, HoL Blocking Eliminating Scheduler (HBES), to improve the performance of MPQUIC in mobile networks. Firstly, HBES provides a fair allocation of aggregated bandwidth for different streams based on their priority. Then, it keeps stream data arriving at the receiver in order by estimating packet arrival time to mitigate HoL blocking and excessive buffer usage. We implement HBES and evaluate its performance in various network scenarios. Experimental results verify the superiority of HBES in reducing stream completion time and buffer occupation over those existing MPQUIC schedulers.**

***Index Terms*— Multipath QUIC, head-of-line blocking, mobile networks, packet scheduling algorithm.**

I. Introduction

SINCE a significant fraction of Internet bandwidth is consumed by web browsing, web page loading time has been a crucial metric to the Internet. Studies show that even a small improvement in web performance can have a significant positive impact in terms of revenue and customer base [1], [2]. Over time, more services are provided as web pages, such as search engines like Google and Bing, online shopping centers like Amazon, and online video platforms like YouTube. With the continuous enrichment of web page functions, the types and sizes of components in web pages are becoming more diverse, which makes it a challenge to optimize web page performance.

Web pages are typically composed of a number of individual objects, and all of them need to be downloaded in order to complete the web page loading. To speed up a web page loading task, Hypertext Transfer Protocol version 2 (HTTP/2) [3] is proposed with features such as multiplexing, stream prioritization, header compressions, and server push. Except for the effort to reduce protocol overhead for more efficient transmission, HTTP/2 brings about significant improvements by supporting the prioritization of all web page objects in order to properly allocate limited network resources to each object according to its priority and dependency. In this way, the crucial objects in the web page rendering process (e.g., HTML files, CSS files, script files, figures with key information, etc.) will be downloaded first or with more network resources. Given the same network capabilities, browsers with proper prioritization settings may finally provide a much better quality of experience for users [4].

In order to further improve the transport performance of HTTP/2 traffic, QUIC [5], an encrypted, multiplexed, and low-latency transport protocol is proposed. HTTP/2 over QUIC aims to break some fundamental limitations of the original version of HTTP/2 over TLS and TCP, including protocol entrenchment, handshake delay, and head-of-line (HoL) blocking [6]. QUIC has shown its advantages by successfully reducing the latency of Google Search responses by 3.6% and rebuffer rates of YouTube playbacks by 15.3% for mobile users. Until now, it has been widely deployed [6] and supported by many major browsers [7].

Although HTTP/2 and QUIC optimize the web page loading process with many methods, the bandwidth limitation of a specific network interface still remains a constraint on the loading speed, which has spawned a technology of concurrently using multiple network interfaces (e.g., Wi-Fi and LTE) of a single device. Such technology is deemed as multipath

Manuscript received 4 April 2022; revised 22 July 2022; accepted 30 September 2022. Date of publication 18 October 2022; date of current version 11 April 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 61972371 and in part by the Youth Innovation Promotion Association of Chinese Academy of Sciences (CAS) under Grant Y202093. The associate editor coordinating the review of this article and approving it for publication was J. Liu. *(Corresponding author: Kaiping Xue.)*

Yitao Xing, Kaiping Xue, Jiangping Han, Jian Li, and Qibin Sun 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 and Jun Lu are 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 Department of Computer and Information Science, Fordham University, Bronx, NY 10458 USA.

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

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TWC.2022.3213638.

Digital Object Identifier 10.1109/TWC.2022.3213638

QUIC extensions. By sending data over multiple paths between a client and a server, QUIC with multipath extensions provides both higher bandwidth and smoother network handover compared with single-path QUIC. Since multipath capability has been proven to be very beneficial and has been adopted by some major software vendors [8], [9], multipath QUIC extensions such as multipath QUIC (MPQUIC) [10], pluginizing QUIC [11] and multiflow QUIC [12] have the potential to be widely used in the foreseeable future. Note that though we focus on MPQUIC in this paper, our work is not necessary to be built on top of any specific multipath QUIC extension implementation because it only utilizes the most common features and can be implemented easily based on different extension versions.

However, extensions like MPQUIC are still prototypes and are inadequate to support complex HTTP/2 traffic in mobile networks. Though it achieves bandwidth aggregation to improve the overall transport performance, two main issues prevent it from reaching its optimal performance. *The first one* is its under-designed packet scheduling method. Highly motivated by multipath TCP (MPTCP) [13], the packet scheduling method of MPQUIC is based on TCP's single bytestream abstraction instead of the multiplexing stream abstraction of HTTP/2 or QUIC. That means that existing MPQUIC cannot fully support the methods that HTTP/2 uses, such as stream prioritization and dependency, to speed up website loading. *The second one* is the heterogeneous and dynamic conditions of mobile networks. Dealing with mobile networks is challenging in many previous studies [14], [15], [16], [17]. In the context of multipath transmission, the out-of-order packets problem is a well-known issue caused by mobile networks. Specifically, out-of-order (OFO) packets may be generated by sending data through multiple paths with asymmetric and ever-changing conditions in mobile networks, leading to HoL blocking and excessive receive buffer occupation. This issue is also known to other multipath protocols such as MPTCP and CMT-SCTP [18]. Though several packet scheduling methods are designed to solve the issues [19], [20], [21], [22], [23], [24], they are not perfectly suitable for MPQUIC because of the stream multiplexing abstraction and are not very effective in dynamic mobile networks either.

To solve the two issues mentioned above, we propose a stream-aware packet scheduler for MPQUIC called HoL Blocking Eliminating Scheduler (HBES), which is able to make priority-based bandwidth allocation for multiplexing streams and execute fine-grained per-packet scheduling based on multipath network conditions. To be specific, HBES adopts the dependency tree structure to handle stream prioritization and decides the number of packets for each stream to send based on a scattered weighted Round-Robin (SWRR) algorithm. Then, HBES schedules each packet according to its estimated arrival time to ensure that packets through different paths arrive at the receiver in order. By doing so, the issues of HoL blocking and excessive buffer occupation can be mitigated. To cope with the fluctuations in mobile networks, HBES tracks the round-trip time (RTT) and throughput of each path, and thus it is able to modify its scheduling decisions when network fluctuation occurs. We implement HBES based on the MPQUIC open-source implementation written in go [25], and evaluate its performance by comparing it with other schedulers in a Mininet-based [26] testbed.

The main contributions of this paper are summarized as follows.

- We propose a stream manager to achieve priority-based bandwidth allocation with the SWRR algorithm, which avoids the HoL blocking among HTTP's multiplexing streams in existing stream managing algorithms. Specifically, SWRR scales down the number of packets each stream can send at a time to prevent a fraction of streams from overusing the send window.
- We propose a stream-aware packet scheduler, called HBES, for MPQUIC, which is designed to estimate the arrival time of each packet and make scheduling decisions accordingly. HBES keeps tracking the RTT and throughput of all paths and modifies its scheduling decisions to stay effective even when network fluctuation occurs in mobile networks. In this way, HBES tries to ensure that packets sent through different paths arrive in order so as to mitigate the HoL blocking caused by OFO packets.
- We implement an HBES prototype in go language and compare its performance with some existing MPQUIC schedulers. To implement HBES, we add a path-level send buffer and modify the dependency tree of the original MPQUIC implementation. We also keep the modifications transparent to the applications that use HTTP. The results show that HBES reduces the two kinds of HoL blocking and optimizes the loading performance of various kinds of web pages, even in asymmetric and dynamic scenarios.

The rest of this paper is organized as follows. In Section II, we introduce HTTP and (MP)QUIC protocols and point out the deficiency of the current MPQUIC design. Then, in Section III, we carefully describe our proposed MPQUIC scheduling algorithm named HBES, followed by some discussion and analysis of HBES in Section IV. We evaluate HBES along with other existing MPQUIC schedulers in a Mininet-based testbed in Section V. And in Section VI, we discuss some relevant scheduling algorithms for different multipath transport protocols. Finally, Section VII concludes our work.

II. Background and Motivation

In this section, we first present an overview of HTTP, QUIC, and MPQUIC. Then, we discuss how multipath extensions like MPQUIC can benefit HTTP traffic in mobile networks and why its performance is currently below expectations.

A. The Hypertext Transfer Protocol and QUIC Protocol

Hypertext Transfer Protocol (HTTP) is an application layer protocol for distributed, collaborative, hypermedia information systems. It has become one of the foundations of data communication for the World Wide Web, providing easy access to abundant network resources for users. Nowadays, HTTP/2 [3] is the most efficient version used by more than 43.3% of the websites [27] and is supported by almost all

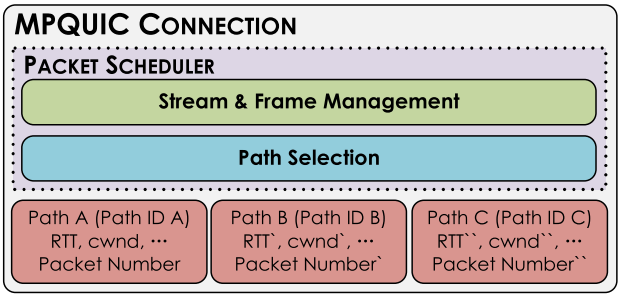
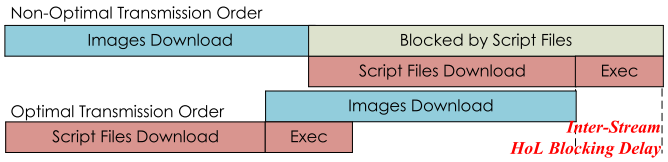
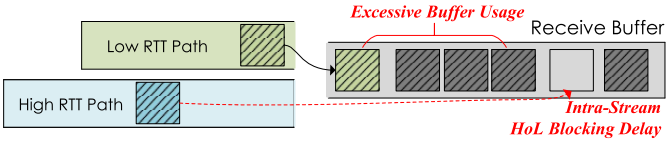


Fig. 1. High-level architecture of Multipath QUIC.



(a) Inter-stream HoL blocking caused by lack of support for prioritization.



(b) Intra-stream HoL blocking caused by heterogeneous network paths.

Fig. 2. Two kinds of HoL blocking that MPQUIC may suffer from.

web browsers because of its ability to speed up page loading. For example, HTTP/2 supports header compression to reduce protocol overhead and allows a server to "push" resources before a client requests them with a server push mechanism.

Additionally, a significant improvement in HTTP/2 comes from stream multiplexing, which aims at solving HoL blocking problem in HTTP/1. Stream multiplexing allows multiple streams of data to reach the receiver in arbitrary order without waiting for a slow stream that is blocking others. And HTTP/2 prioritization allows a client (e.g., a browser) to assign a priority for each stream, which can be used to instruct the server to allocate limited network capacity for different streams accordingly. However, despite all these efforts, HTTP/2 hosted on TCP may still suffer HoL blocking if any of the TCP packets are delayed or lost.

To further optimize the performance of HTTP traffic, QUIC [5] is designed to break some fundamental limitations of the TLS/TCP ecosystem. QUIC uses a UDP-based lightweight data-structuring abstraction, *streams*, which are multiplexed within a single connection, so that the loss of a single packet blocks only the streams with data in that packet [6] rather than all streams. Compared with HTTP/2 over TLS and TCP, HTTP/2 over QUIC eliminates HoL blocking delays caused by TCP's single-bytestream abstraction and can be rapidly improved and easily deployed by moving from kernel space to user space. Considering its great advantages, the IETF's HTTP and QUIC Working Groups have jointly decided to call the HTTP mapping over QUIC "HTTP/3", making it a worldwide standard in advance [28].

B. Multipath Extension for QUIC

One missing feature of the current design of QUIC is the ability to exploit the multiple paths that potentially exist between the two hosts. Given the fact that mobile devices nowadays are always equipped with multiple network interfaces (e.g., LTE and Wi-Fi), MPQUIC [10], as one of the multipath extensions for QUIC, is designed to aggregate bandwidth from those interfaces and also to achieve smoother network handovers. The architecture of MPQUIC is shown in Fig. 1.

MPQUIC creates one path with a unique *Path ID* over each interface, and packets are selected and transmitted over different paths by a *packet scheduler*. Each path has its own packet number space. Thus, each packet can be identified with the combination of *Path ID* and packet number in its public header, which can be used to generate ACKs for reliable transmission. Except for the header, a QUIC packet consists of one or more frames carrying control information or stream data. Frames carrying stream data as well as a stream identifier (*Stream ID*) and a byte offset are named *STREAM frames*, and a receiver is able to reorder *STREAM frames* from different paths only with the information attached to them.

C. Current MPQUIC Design Faces Challenges

Although MPQUIC can actually improve overall performance compared with single-path QUIC by providing higher bandwidth, its performance of supporting HTTP traffic (e.g., web page load) is not yet optimized. To be specific, the de facto *packet schedulers* in MPQUIC are borrowed from MPTCP directly. Thus, these schedulers cannot respond to stream priority properly and may lead to *inter-stream HoL blocking*. For example, when loading a web page, current MPQUIC schedulers may choose to send low-priority streams, such as some unimportant images, first. However, the web page will not show any of them since the crucial CSS or JavaScript files that are actually blocking the webpage rendering process have not yet been transmitted, as shown in Fig. 2a. Since the *inter-stream HoL blocking* has been proven to be an obstacle to satisfying performances of jobs such as webpage loading [29], the ability to handle the multiplexing and prioritized streams is a must for MPQUIC schedulers.

Besides, current schedulers may also cause *intra-stream HoL blocking* by sending packets carrying *STREAM frames* from one stream through several asymmetric paths. As shown in Fig. 2b, when packets carrying frames with lower byte offset are delayed on paths with higher RTT, HoL blocking occurs and may lead to extra reordering delay and excessive receive buffer usage. Coping with paths with heterogeneous characteristics is a known issue for multipath transport protocols, and designing better packet schedulers is an effective approach to solve it [19], [20], [21], [22]. However, since these schedulers are not designed for MPQUIC featuring stream multiplexing, using them directly is not a wise decision. In addition, they may not be able to stay effective in mobile networks where network conditions of paths are asymmetric and fluctuating. As a result, in order to improve the performance of HTTP

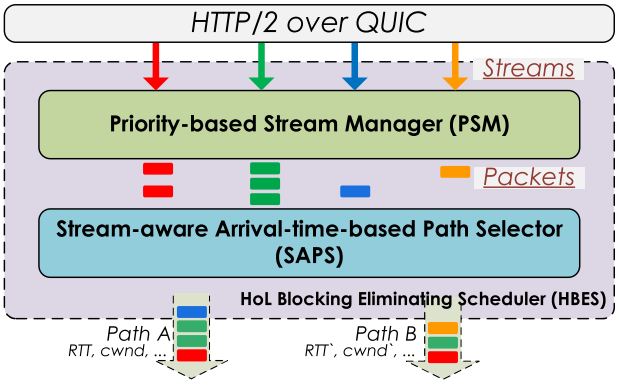


Fig. 3. The framework of HBES, consist of two schemes named PSM and SAPS.

traffic over MPQUIC in mobile networks, a new packet scheduler needs to be developed.

D. Design Objectives

To sum up, MPQUIC packet schedulers have to deal with *the out-of-order issue of multipath transmission* and handle *HTTP's multiplexing and prioritized streams*. Simultaneously dealing with the two issues becomes a unique problem with carrying HTTP traffic with MPQUIC.

In this paper, we propose a new MPQUIC scheduler that achieves the following two goals at the same time.

- Priority-based network resources allocation among streams without any *inter-stream HoL blocking*.
- Reducing the number of out-of-order packets caused by heterogeneous wireless network paths with changing characteristics to avoid *intra-stream HoL blocking*.

III. HoL Blocking Eliminating Scheduler: Design and Implementation

A. Overview

The HoL Blocking Eliminating Scheduler (HBES) consists of two schemes, namely, a Priority-based Stream Manager (PSM) and a Stream-aware Arrival-time-based Path Selector (SAPS). In general, PSM is designed to decide which stream to transmit data and how many packets it can send according to its priority in order to avoid *inter-stream HoL blocking* caused by non-optimal network capacity allocation among streams. And then, for each packet, SAPS selects a path in which the arrival time of this packet is the lowest. In this way, *intra-stream HoL blocking* can be mitigated when packets arrive in order. SAPS also considers network fluctuation for effectiveness in ever-changing mobile networks. The framework of HBES is shown in Fig. 3.

In order to handle HTTP/2 stream prioritization, PSM adopts a dependency tree structure defined in RFC 7540 [3]. Fig. 4 is an example of a dependency tree. Each node in the tree represents a stream created by the MPQUIC connection. A client assigns a priority to each stream in the dependency tree, and streams can be given an explicit dependency on another stream, which makes them children and parent.

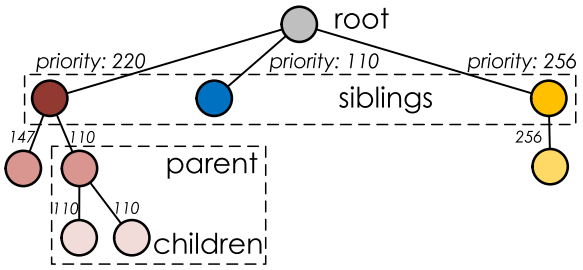


Fig. 4. An example of the HTTP/2 dependency tree. Each node represents a prioritized stream.

A stream is only allowed to transmit after its parent completes transmission, and its priority can be used to determine the relative proportion of available resources that are assigned among its siblings (the streams that have the same parent). Specifically, a stream that is not dependent on any other stream is a child of the root.

Weighted Round-Robin (WRR) is a widely used algorithm to manage the prioritized streams and achieve priority-based network resource allocation. A common way [30], [31] to implement a WRR-based stream managing algorithm is letting each stream send a specific number (equal to its priority) of packets in a round, then continues with another round of execution. But such an implementation is not optimal because it still causes HoL blocking even with the multiplexing and prioritization features. The reason is that streams can be blocked if other streams have consumed the sending window (swnd) before they are allowed to send anything. And the order to traverse the streams is not related to the streams' priority, as the RFC states, which can lead to the blocking of high-priority streams. This issue happens more frequently when the streams' priority numbers are relatively large (e.g., from 110 to 256 in Chrome [29]).

For PSM, we design a better stream managing algorithm called scattered WRR (SWRR) to mitigate this blocking issue. Based on the streams' priority, SWRR scales down the number of packets a stream can send in each round (or in each *scheduling cycle*), leaving the send windows available and letting more streams send packets in each round. In other words, with SWRR, the send windows will not be exhausted by a fraction of streams that are luckily at the front.

After PSM selects a stream to send a packet, SAPS decides the path through which the packet should be sent. Basically, SAPS tries to eliminate the HoL blocking caused by OFO data frames by sending each packet according to its arrival time. For more precise arrival time estimation, SPAS needs to track the current RTT, throughput, and send buffer occupation of each path. Then, SPAS estimates the arrival time T of a packet consisting of its queuing time in send buffer and its inflight time, though it may schedule packets to a currently unavailable path. For example, when SPAS estimates that a packet will arrive at the receiver through path l in $T_l = 20\ ms$ and through another path k in $T_k = 40\ ms$, then the packet should be delivered via path l, even though it is not available due to limited congestion window. In this situation, SAPS believes that though the packet will be queued in the send buffer of path l and wait for a period, it will still arrive earlier.

We present detailed descriptions of these two schemes in the rest of this section. In this paper, we assume that all streams are client-initiated, and we use odd IDs [6] to identify the streams for presentation simplicity. Note that our schemes do not rely on the stream IDs, so they can be deployed on both client and server sides, wherever an endpoint must handle prioritized streams over multiple network paths.

B. Priority-Based Stream Manager (PSM)

To mitigate the *inter-stream HoL blocking* and further improve the performance of HTTP traffic over MPQUIC, the ability to handle stream prioritization and dependency is a necessity, and PSM is designed for this purpose. First, PSM creates a dependency tree every time when an MPQUIC connection starts, and the root of the tree contains a virtual stream with *Stream ID* 3 (stream 1 is a crypto stream and is handled separately with the highest priority). When a stream is initiated, it is added to the tree. If the stream depends on another stream, it will be a child of that stream. Also, an independent stream is a child of the root.

Then, PSM selects a stream to send data using a modified WRR algorithm called scattered WRR (SWRR) (in Algorithm 1). Firstly, PSM decides how many packets a stream can send in a scheduling cycle and assigns a $token$ to each stream with SWRR algorithm. Then PSM visits the children streams of the root in the order they are established and sends $token$ number of packets of each visited child.

To show the difference between the original WRR and SWRR, we first describe the blocking issue that WRR may cause. WRR traverses all the children (or streams) of a parent and allows each of them (e.g., stream i) to send $priority_i$ numbers of packets. Then it continues with another round of execution. Here, $priority_i$ is an integer number between 1 and 256 (inclusive) given by an HTTP/2 client (or a browser). Such a design follows RFC's [3] suggestions for prioritized streams, but it still causes HoL blocking since a fraction of streams with large priority numbers can exhaust the sending window (swnd) before others. Note that low-priority streams can have large priority numbers (e.g., 110 is the lowest in Chrome [29]). For example, consider this scenario: There are only 32 packets that the MPQUIC connection can send due to the limited window size. And a stream i with $priority_i = 32$ may be allowed to send 32 packets, blocking a stream j with $priority_j = 128$ which only has 2 packets to send.

A better strategy here is to let stream i send 1 packet and let stream j send 4 packets according to the ratio of their priority, which is the basic idea of SWRR. With SWRR, the $token$ is no longer directly equal to the priority number. Instead, it is smaller to avoid the blocking issue. Specifically, as shown in `SetRescaledToken` in Algorithm 1, PSM scales down the $priority$ of all siblings (streams that have the same parent) to a smaller range according to a rescale factor r[1] ($token$ is set to 1 if it equals 0). By doing so, the number of packets sent in one scheduling cycle is much smaller, so the send window is less likely to be exhausted before some streams even have the chance to be visited by PSM. That is the way PSM uses to mitigate the inter-stream HoL blocking.

Algorithm 1 Priority-Based Stream Managing

```
1  Function ScatteredWRR(node):
2      if for all the siblings, token = 0 then
3          SetRescaledToken(node.parent)
4      stream = node.stream
5      if node.token > 0 then
6          if stream has data to send then
7              selected = stream
8          else
9              selected =
               ScatteredWRR(node.nextChild)
10     if selected ≠ nil then
11         node.token -= 1
12     if selected = nil or node.token = 0 then
13         node.parent.nextChild = node.nextSibling
14     return selected
15 Function SetRescaledToken(node):
16     find the maximal priority M of all children
17     /* r is a given parameter */
18     γ = M/r
19     foreach child of the node do
20         /* each stream can send a token
           number of packets per round */
21         child.token = [node.priority/γ]
22 Function Main:
23     all the nodes are initiated with token = 0
24     while there is active node in the tree do
25         stream = ScatteredWRR(root.nextChild)
26         send one packet of the stream
27         foreach node in the tree do
28             if node.stream finishs sending then
29                 deactive node
```

Overall, starting from the root, the children are allowed to send data in packets and consume their tokens one by one. After all, streams run out of their sending opportunities, PSM resets them for a new scheduling round.

Note that PSM ensures that each stream gets certain "opportunities" to send data in packets but does not guarantee that the packets only contain its *STREAM frames*. This is because, in QUIC protocol, certain frames carrying control information or the frames being retransmitted should be prioritized over *STREAM frames*. Thus, they are encapsulated in packets first, and the remaining space is for *STREAM frames* to use. As a result, the amount of stream data that a packet can carry is uncertain. There may be two ways to make sure that a stream can always send a fair share amount of data. The first one is to additionally generate small packets to carry control information and retransmissions, which is not efficient due to increasing header overhead. The other option is to attach its

[1]We will discuss how r works in practical use in Section IV.

TABLE I
SYMBOLS AND MEANINGS

Acronyms	Meanings
tp	throughput
qt	queuing time
ft	flight time
qs	queue size
ps	packet size
T	packet arrival time
$cwnd$	congestion window size
rtt	round-trip time
r	rescale factor

Algorithm 2 Path Selecting Algorithm of SAPS

Input : path set $\mathbb{P}$, packet of stream i,
stream i data was previously sent on path p.
Output: selected path ($path$).

Function `PathSelecting(`$\mathbb{P}$, i, p`)`**:**
  if *cwnds of all paths are full* **then**
    return **nil**
  for $j \in \mathbb{P}$ **do**
    calculate arrival time using $T_j(rtt_j)$
  find path $path = \arg\min_j T_j(rtt_j)$
  return $path$

leftover data to another packet with remaining space, which is not ideal either because a lost packet may affect multiple streams in this situation. As a result, PSM allocates sending opportunities instead of a certain amount of data to make it simple and flexible.

C. Stream-Aware Arrival-Time-Based Path Selector (SAPS)

In order to mitigate the *intra-stream HoL blocking*, SAPS is designed to make frames from the same stream arrive in order at the receiver side. Algorithm 2 presents the overall working process of SAPS, and TABLE I summarizes all the acronyms used in this paper. For this purpose, SAPS estimates the arrival time of each packet according to current RTT and queuing delay and then schedules the packet for the path with the shortest arrival time. In other words, let T_i denotes the arrival time of a packet sent via path i, then SAPS should schedule the packet for path k which satisfies $k = \arg\min_i T_i$.

T_i consists of two parts, which are the time that a packet wait in the send buffer of path i (i.e., queuing time, qt_i), and the time that it is inflight (i.e., flight time, ft_i). That is,

$$\begin{aligned} T_i &= qt_i + ft_i \\ &= \frac{qs_i + ps}{tp_i} + \frac{rtt_i}{2}, \end{aligned} \tag{1}$$

where qs_i, rtt_i and tp_i denote the queue size of the send buffer, the smoothed RTT, and throughput of path i, respectively. And ps denotes the size of a packet, which is not always the same actually. However, since MPQUIC tries to send the largest possible packets for higher efficiency, it is safe to use a fixed ps for all packets, which is the max packet size defined in QUIC protocol (1450 bytes in implementation). And the throughput can be estimated by

$$tp_i = \frac{cwnd_i}{rtt_i} \times ps. \tag{2}$$

Since T_i is a function of rtt_i, we can present it in the form of $T_i(rtt_i)$ as shown in Algorithm 2.

In this way, SAPS may not always try to fill the congestion windows of all available paths. Instead, it may keep building up the queue in the buffer of a path with lower RTT and finally switch to another path when the queuing time is too long. And at the same time, SAPS keeps tracking the current RTT and queuing data size of all paths in order to adjust its scheduling decisions accordingly in changing network scenarios. Note that though the scheduling decisions for the queued packets may be outdated due to abrupt network changes, such mistakes will not accumulate or significantly negatively affect the overall performance. That is because SAPS will pause when all paths' congestion windows are unavailable, and current scheduling decisions only affect the limited number of scheduled packets. And when SAPS continues to schedule packets, it makes decisions based on up-to-date network conditions.

In Section V, we prove that SAPS work effectively in a wide range of scenarios that are common in mobile networks.

D. Implementation

Our HBES is implemented based on the open-source code of MPQUIC [10], PStream [31] and SA-ECF [30]. PStream and SA-ECF build the dependency tree structure on top of MPQUIC, which can be used for PSM's SWRR algorithm with some modifications and bug fixes.

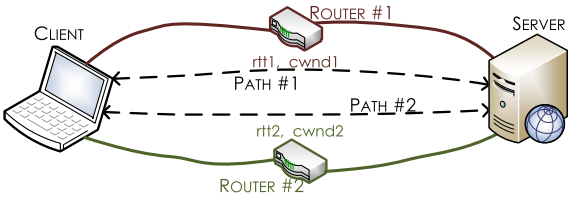
To implement SAPS, we have to implement a path-level send buffer in addition to the existing "main send buffer". In the current MPQUIC implementation, there is no send buffer for each path. Thus, a packet will be immediately sent out after it is derived from the main send buffer and is scheduled to a certain path. But SAPS sends packets through the path with the shortest packet arrival time, even though the path may not be available currently. That means that some packets may have to wait for the path to be available in the path-level send buffer. In the implementation of SAPS, we maintain a path-level send buffer to store the payloads of the packets. Because MPQUIC sends packets with strictly monotonically increasing packet sequence numbers, the payloads are sealed with the packet headers when they are finally sent out. Note that such a structure can also support the implementations of some schedulers that are initially designed for MPTCP to improve MPQUIC.

IV. DISCUSSION ON HBES

In this section, we discuss how HBES benefits HTTP/2-like traffic carried by MPQUIC in heterogeneous networks.

A. Reducing Intra-Stream HoL Blocking

When packets arrive at the receiver out-of-order, intra-stream HoL blocking happens. It is such a common issue



(a) The topology of our testbed.

(b) One of the most common mobile network scenarios.

Fig. 5. Network topology for performance evaluation in our Mininet-based testbed.

that some naive multipath schedulers take it for granted. However, in mobile networks where devices may be buffer limited, these out-of-order may be dropped if the receive buffer is full, leading to severe performance degradation. To illustrate how HBES mitigates this issue, we discuss and present the sufficient buffer size needed to prevent MPQUIC's overall throughput from decreasing for different schedulers. The following discussion is based on one of the most common topologies in mobile networks (shown in Fig. 5a), which contains two paths denoted by path 1 and 2. The RTT, congestion window size, and throughput of path i are denoted by rtt_i, $cwnd_i$, and tp_i, respectively.

A naive scheduler named Round-Robin (RR) ignores the different characteristics of subflows and simply sends one packet on each subflow in turn. Such a scheduler requires a large buffer size on the receiver side. Specifically, assuming $rtt_2 > rtt_1$, the receive buffer should be large enough to store all the packets from path 1 before the packets from path 2 arrive. That is, the required buffer size should be larger than

$$tp_1 \times \frac{rtt_2}{2} = cwnd_1 \times \frac{rtt_2}{2 \times rtt_1}.$$

That means that the needed buffer size is proportional to the capacity of path 1 and the RTT ratio of the two paths. With the growth of link capacities, such a scheduler may take up much memory space and even consume more in heterogeneous mobile networks.

Lowest-RTT-First (LRF) scheduler takes heterogeneity into consideration. It will fill the congestion window of the subflow with the lowest RTT before using other subflows. For small streams, LRF guarantees no out-of-order packets, but when the streams are large enough, LRF has to use the subflow with higher RTT, which will, again, cause intra-stream HoL blocking. Since we are discussing the sufficient buffer size needed, we can assume that $rtt_2 > 2 \times rtt_1$ in a heterogeneous network. In this case, the first $cwnd_1$ packets from path 1 arrive in order and will be passed to applications directly. But before packets from path 2 arrive, there are more packets from path 1 arrive earlier. As a result, the needed buffer size for LRF should be larger than

$$tp_1 \times (\frac{rtt_2}{2} - rtt_1) = cwnd_1 \times (\frac{rtt_2}{2 \times rtt_1} - 1).$$

This result illustrates that LRF only reduces a limited amount of receive buffer occupation than RR, and the buffer needed is also proportional to the link capacity and the RTT ratio. Note that SA-ECF behaves the same as LRF when there is much data left to send. Thus, it also requires a large receive buffer to achieve the expected performance.

HBES, different from all the MPQUIC schedulers mentioned above, will not create OFO packets theoretically. Because by skipping some subflows, it always sends packets through the subflow that enables the packets to arrive as early as possible. However, it cannot be denied that in practical use, it will cause some OFO packets because the measurements of link capacity and RTT can be inaccurate. But the number of OFO packets can be largely reduced even in highly heterogeneous network conditions. Our evaluation in Section V verifies our analysis.

TABLE II
RESCALING CHROME'S PRIORITY VALUES WITH $r = 10$

Priority Level	Lowest	Low	Normal	High	Highest
Typical Value	110	147	183	220	256
Rescaled Value	4	6	7	9	10

B. Reducing Inter-Stream HoL Blocking

The inter-stream HoL blocking issue is a unique problem that multipath QUIC encounters when dealing with multiple streams simultaneously. To mitigate this issue, WRR algorithm is used by some MPQUIC schedulers such as SA-ECF [30]. However, as we discussed in Section III-B, the way browsers set stream priority may lead to a large scheduling cycle which may finally cause the blocking of short streams. For example, Chrome's priority values are from 110 to 256 [29], and thus hundreds of packets will be scheduled within one cycle. Even with the per-stream swnd constraint of HTTP/2, blocking can still happen when there are many streams. And the same problem happens to Safari and Firefox too. As a result, HBES uses SWRR algorithm to rescale the priority to a smaller range with a rescale factor r. TABLE II presents the rescaled value of Chrome's priority values, which are calculated following the `SetRescaledToken` function in Algorithm 1. For example, in the case in TABLE II, PSM firstly finds the maximal priority M of all streams ($M = 256$), and calculates $\gamma = M/r = 25.6$. Finally, a stream with the lowest priority gets a rescaled value equaling $[110/25.6] = 4$.

We now estimate the period that a short high-priority stream has to wait before it can actually be sent with the original WRR or SWRR algorithm. We assume that MPQUIC needs to send n streams simultaneously, and each stream is allowed to send m packets within one scheduling cycle. Note that m is usually limited by the flow control window of the streams

according to [25] instead of the large priority values. Since the priority values do not indicate the transmission order, the high-priority stream may be randomly sent at the X^{th} place with uniform distribution. That means that the stream has to wait for $(X-1)\times m$ packets from other streams to be sent first. The expected number of those packets is

$$\mathbf{E}((X-1)\times m) = \sum_{x=1}^{n} \frac{(x-1)\times m}{n} = \frac{m\times(n-1)}{2}.$$

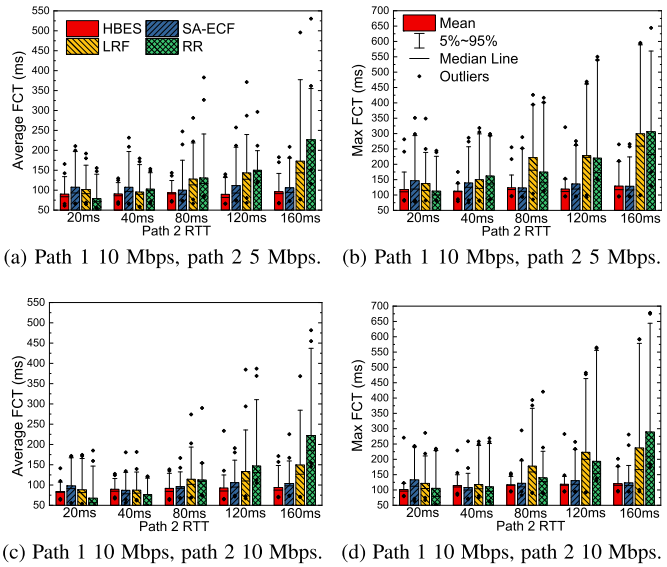
When the number of concurrent streams grows higher, the high-priority stream has to wait for more linearly growing packets. In some cases, short high-priority streams, which are expected to finish sending in the first RTT, must wait for another RTT or even more RTTs. With SWRR algorithm, the number can be decreased to $\frac{m'\times(n-1)}{2}$ where m' is the rescaled priority value of the streams. In current MPQUIC implementation [25], $m \approx 11$, while with SWRR, m' can be 4 when choosing $r = 10$. Thus, the expected waiting time of the high-priority stream can be reduced by over 60% theoretically.

To be of practical use, the rescale factor r should be chosen according to the traffics that MPQUIC carries. When the number of concurrent streams is large, a smaller rescale factor can reduce the probability of inter-stream blocking. But if it is too small, the difference in the bandwidth obtained by high-priority streams and low-priority ones will be reduced. In our evaluation, we show that SWRR algorithm with $r = 10$ can effectively mitigate the inter-stream blocking issue while providing differentiated services for multiple streams at the same time.

V. Performance Evaluation

A. Experimental Setup

For performance evaluation, we implement HBES in go language and build a testbed based on the Mininet virtual machine offered by De Coninck et al. [25] with OLIA [32] congestion control algorithm inside. In general, in order to evaluate HBES in the mobile network scenario, we build a testbed with two paths of widely varying network conditions (e.g., RTT, bandwidth, etc.) as shown in Fig. 5a to simulate a prevalent mobile network scenario (Fig. 5b) where the network capability and RTT may significantly change with issues like user movement [33], bufferbloat, etc. Such a topology is common in practical mobile networks, and it is also widely used in other papers [10], [19], [30], [31], [34] for evaluation. Note that the RTT we set for each path is not static in all the experiments. Instead, it is the minimum RTT of the path, and actual RTT may significantly vary due to bufferbloat, which is the same as that in the actual mobile networks. Then, a client generates various kinds of HTTP/2 requests (called workflows in this section) to a server over two paths representing the LTE and Wi-Fi paths which are widely used by mobile devices. The network conditions of the two paths are set from totally symmetric to highly asymmetric in a wide range in order to cover different network scenarios that mobile users may encounter in real networks. Except for HBES, we also evaluate



(a) Path 1 10 Mbps, path 2 5 Mbps. (b) Path 1 10 Mbps, path 2 5 Mbps.

(c) Path 1 10 Mbps, path 2 10 Mbps. (d) Path 1 10 Mbps, path 2 10 Mbps.

Fig. 6. The average and maximal FCT of 5 small streams with various network conditions of paths.

SA-ECF, Lowest-RTT-First (LRF), and Round-Robin (RR) for comparison.

Note that to simulate the web page load processes in the real network, before actually generating test workflows, the client will request for an HTML file in advance. And we start the flow completion time (FCT) recording after receiving the HTML file. In this way, HBES and other schedulers have preliminary estimations about the RTT and throughput of paths.

We conduct two groups of experiments, namely, stream-level evaluation and the evaluation on typical web page loading workflows. We carefully describe them and analyze the results in the next part of this section.

B. Stream-Level Evaluation

In this group of experiments, we generate traffic of short, long, and mixed-size streams over a wide range of path conditions to clearly illustrate the performance of HBES with different workflows.

1) HBES Reduces FCT of Small Streams: In this experiment, a client generates five requests for five 20 KB files with the same priority from a server over two paths, and each test is repeated several times. Path 1 has 10 Mbps bandwidth and 20 ms RTT, while the RTT and bandwidth of path 2 vary in a wide range (from 20 ms to 160 ms). The detailed settings and the results are shown in Fig. 6.

The reasons why we use five 20 KB files as representatives of small stream workflows are as follows: 1) There should be enough data to send on both paths to reveal the difference among schedulers. And since the initial cwnd of each path is roughly 40 KB (according to current MPQUIC implementation), we set the total amount of data as 100 KB; 2) There should be more than one scheduling cycle to show the effect of the stream managing functions of HBES and SA-ECF. Thus, each stream gets 20 KB of data to send, which is larger than the initial stream flow control window (16 KB).

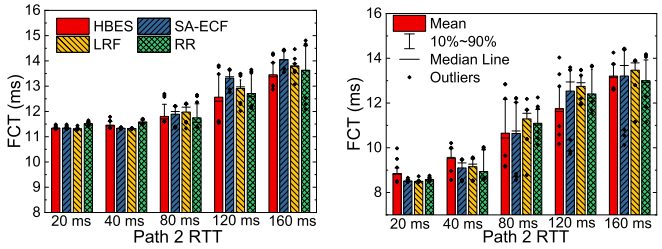
We first focus on the average FCT of the five streams, which is presented in Fig. 6a and 6c. Obviously, when the two paths are of the same RTTs (20 ms for both paths), all schedulers perform well, and RR outperforms other schedulers in this ideal network condition. This is because RR effectively aggregates bandwidth by sending data on both paths, while others prefer to send data on one path with a relatively smaller RTT estimation. As the asymmetry increases, the results illustrate that well-defined HBES and SA-ECF still achieve low FCT while LRF and RR suffer from high FCT, which means that though the designs of HBES and SA-ECF are different, they both can effectively mitigate HoL blocking. Besides, by comparing Figs. 6a and 6c, we found out that although the increment of total bandwidth from 15 Mbps to 20 Mbps reduces the mean and median average FCT of all schedulers, the overall performance improvement may be blurred by the asymmetric paths. One of the reasons is that in order to provide higher bandwidth, schedulers send more data to path 2, but the gain brought by aggregated bandwidth is offset by HoL blocking delay. In a word, HBES and SA-ECF bring significant benefits for small streams whose FCT is relatively small, and HoL blocking delay may account for a large portion of it. We can also state that HBES performs slightly better than SA-ECF in asymmetric scenarios, according to the results.

We also present the maximal FCT of the five streams in Figs. 6b and 6d, which can also be treated as the page loading completion time when all streams finish delivery. These two figures illustrate that the performance of LRF and RR is even worse because they are not aware of multiplexing streams and cannot properly allocate network capacity among streams. In particular, LRF and RR may send a stream exclusively until it is limited by a per-stream send window of HTTP, which leads to a short FCT for that stream, but a much longer FCT for other streams since they have to wait or be sent on path 2 with high RTT. On the contrary, HBES and SA-ECF always provide low maximal FCT by ensuring each stream's fair share of network capacity.

2) HBES Reduces Receive Buffer Usage of Long Streams: In this experiment, a client requests two large 10 MB files with the same priority from a server. The topology and path characteristics are the same as above, and the RTT of Path 1 is 20ms. The results are shown in Fig. 7 and Fig. 8. Fig. 7 shows that HBES still outperforms other schedulers in asymmetric network scenarios by mitigating HoL blocking delay at the tail of a stream, and the most obvious improvement brought about by HBES is the low receive buffer usage for long streams, as shown in Fig. 8.

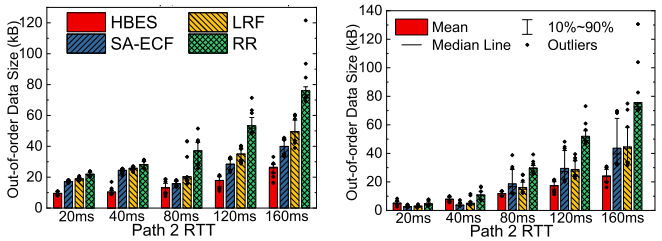
We collect the size of OFO data stored in the receive buffer by recording the bytes offset of *STREAM frames*. Noted that though there may be different approaches to implementing MPQUIC, we make conservative estimations on buffer usage by realistically assuming that a receiver will retrieve all in-order data and empty its buffer as soon as possible.

Fig. 8 illustrates the superiority of HBES in reducing OFO data size. In other words, it's in saving receive buffer usage. This is because HBES is designed to keep data arriving in order all the time, which can mitigate HoL blocking at the end of stream transmitting and reduce OFO data in the middle



(a) Path 1 10 Mbps, path 2 5 Mbps. (b) Path 1 10 Mbps, path 2 10 Mbps.

Fig. 7. The average FCT of long streams under various network scenarios.



(a) Path 1 10 Mbps, path 2 5 Mbps. (b) Path 1 10 Mbps, path 2 10 Mbps.

Fig. 8. The average size of OFO data stored in receive buffer within flow completion time.

of transmitting. By contrast, SA-ECF functions well at the end of stream transmitting but has similar behavior to LRF when a stream still gets much data to send (i.e., in the middle of transmitting). As a result, it also suffers excessive buffer usage in asymmetric network scenarios. To sum up, HBES has the ability to better support buffer-limited devices in asymmetric mobile networks, preventing them from suffering unnecessary retransmissions due to buffer overflow.

3) HBES Mitigates Inter-Stream HoL Blocking Issue: In this experiment, a client requests 10 128 KB files and a 5 KB file with priority set to 110 and 255, respectively. The bandwidth of both paths is 10 Mbps, and the RTT of path 1 is 20 ms. In order to show how small but important streams can be overwhelmed by low-priority streams and how HBES solves this problem, we focus on the FCT of the high-priority small stream, which is presented in Fig. 9. We note the full-functioned HBES (i.e., HBES with SWRR algorithm) as *HBES-SWRR*, and note HBES with original WRR algorithm as *HBES-WRR*.

The result shows that in asymmetric network scenarios if a scheduler does not care about the priority of streams (such as LRF or RR), the important stream may suffer long FCT, which is about 1.7 times higher than the median. However, HBES with the original WRR algorithm also delays the FCT of the high-priority stream in the cases where the high-priority stream is scheduled after some other low-priority ones (the scheduling order is random and not priority-related), which also builds up a long queue before high-priority data. Though SA-ECF also uses WRR, it seems that the blocking issue is less severe. This is because SA-ECF is designed to send data of large streams to the path with longer RTT and thus reduces the amount of data that has to be sent before high-priority data. In general, HBES (with SWRR) successfully reduces the FCT of the most

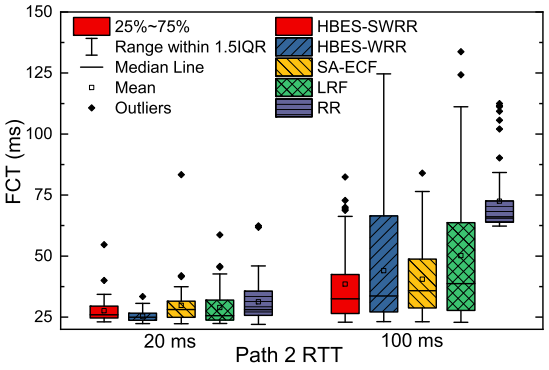
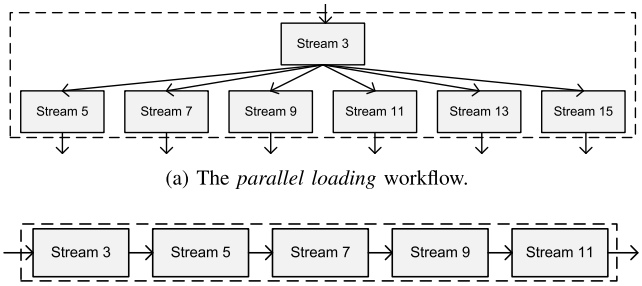


Fig. 9. FCT of the small but high-priority stream using full-functioned HBES (with SWRR) and other schedulers.



(a) The *parallel loading* workflow.

(b) The *serial loading* workflow.

Fig. 10. Two kinds of typical workflows that make up dependency trees of common web pages.

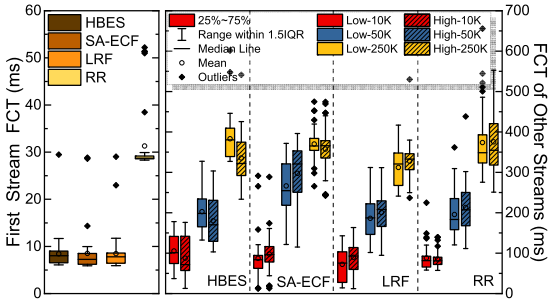


Fig. 11. FCT of all streams in "parallel" workflows.

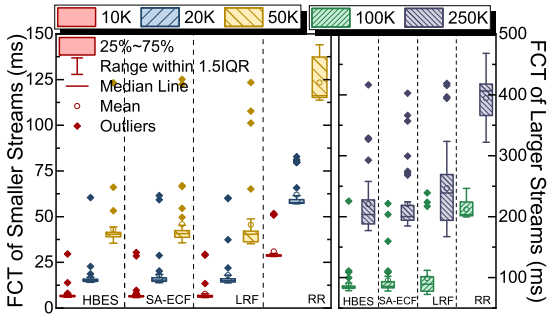


Fig. 12. FCT of all streams in "serial" workflows.

important stream by shortening the scheduling cycle, which means that in complex tasks with many concurrent streams, high-priority streams can complete transmission as soon as possible.

C. Typical Workflows Evaluation

In this section, we compare the performance of HBES with other schedulers based on two kinds of typical workflows, namely, *parallel loading* and *serial loading*. In this paper, the term *parallel loading* refers to a kind of web page loading process which is composed of several streams depending on the same stream (Fig. 10a). The term *serial loading* refers to another kind of process that several streams are loaded one by one (Fig. 10b). These two kinds of loading workflow are regarded as typical because they can make up most of the web page dependency trees generated by different browsers [29]. Thus, evaluation based on them can reflect how schedulers work in practical use and is much easier to analyze than loading complex real web pages. In the following experiments, the bandwidth of both paths is set to 10 Mbps, and the RTT of the two paths is set to 5 and 50 ms, which simulates the latency of a good Wi-Fi channel and an average 4G channel [35], respectively.

1) HBES Speeds up High-Priority Streams Loading: In this experiment, the client requires a parallel loading workflow consisting of one 10 KB files followed by 6 different files. The 6 files are divided into two priority level (priority is set as 110 and 256 respectively) and three sizes (10 KB, 50 KB and 250 KB). Timing starts from the first stream, the FCT of all streams are presented in Fig. 11. The figure is split into two parts for easy viewing.

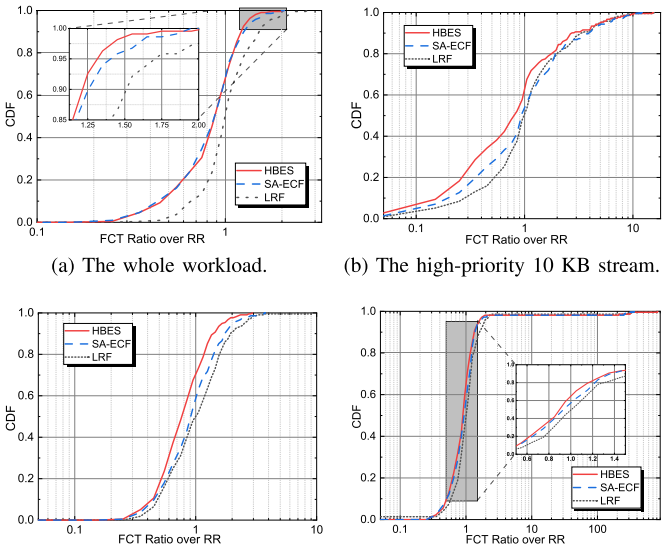
Fig. 11 illustrates that LRF outperforms others in terms of the completion time of the whole loading process because the total size of all streams is so large that the aggregated bandwidth becomes the biggest influencing factor instead of intra-stream HoL blocking delay. However, LRF, as well as RR, ignores the different priority of streams, which is undesired. And surprisingly, SA-ECF fails to react to streams of different priority — for 10 and 50 KB streams, the FCT of the high-priority ones is longer than that of the low-priority ones due to the blocking issue caused by WRR algorithm that we mentioned in Section III-B. On the contrary, HBES shows its ability to allocate network resources among streams properly. The result shows that HBES successfully shortens the FCT of high-priority streams of all sizes. As a result, when downloading several web page objects in parallel, HBES is able to speed up the important high-priority object loading to optimize the web page rendering process.

2) HBES Shortens Stream FCT of Serial Loading Workflows: In this experiment, the client requests a serial loading workflow consisting of 5 different files of 10, 20, 50, 100, and 250 KB, which are downloaded one by one. Timing also starts from the first stream, and the FCT of all streams is presented in Fig. 12.

In this experiment, HBES and SA-ECF show superiority in speeding up stream transmitting in asymmetric mobile networks. For smaller streams (i.e., 10, 20, and 50 KB streams), only RR suffers longer FCT because it is the only algorithm that will use the path with longer latency, while for other streams, LRF starts to send data on the path with long latency.

TABLE III
DOMAINS OF THE INFLUENCING PARAMETERS

Parameters	Path #1	Path #2
Queuing Delay (ms)	0 - 30	0 - 30
RTT (ms)	0 - 20	0 - 200
Link Capacity (mbps)	5 - 20	5 - 20



(a) The whole workload. (b) The high-priority 10 KB stream.

(c) The high-priority 50 KB stream. (d) The high-priority 250 KB stream.

Fig. 13. The FCT ratio of different schedulers over RR.

On the contrary, HBES and SA-ECF still achieve shorter FCT because they choose to skip the long-latency path. Although HBES and SA-ECF behave similarly in this experiment, their scheduling decisions are different. Specifically, SA-ECF chooses to wait until the preferred path is available or act like LRF, while HBES can still schedule packets to the currently unavailable path thanks to an extra send buffer structure. This difference leads to performance distinction as shown in previous experiments (Fig. 8), but it is not obvious in this experiment.

3) Overall Performance Evaluation Across the Parameter Space: Though the results shown above have validated the advantages of HBES in some typical network scenarios, the network conditions in the real wireless network are random and complicated. To fully evaluate HBES and other MPQUIC schedulers, we need to conduct experiments across the whole parameter space. As a result, we adopt the WSP algorithm [36] to generate a spacing-filling design that fills the parameter space evenly with over 200 parameter sets (the domains of the parameters are shown in TABLE III). By going through the whole parameter space, we can evaluate HBES and other schedulers thoroughly with over 200 random network scenarios. We adopt the same *parallel loading workflow* in this experiment as we describe above. Then the FCT of the whole workflow represents schedulers' performances of transmitting a trunk of data, and the FCT of high-priority streams illustrates how schedulers deal with streams of different priority.

The results are shown in Fig. 13. We divide the FCT of HBES, SA-ECF, and LRF by that of RR (denoted by *FCT ratio over RR*) in the same network scenario to show the performance improvement of these schedulers. The lower the FCT ratio is, the better a scheduler performs in this case.

As shown in Fig. 13a, HBES and SA-ECF shorten the overall FCT of the workload by 70% of all cases, while LRF does not show significant improvement. This result illustrates that HBES, as well as SA-ECF, does well in reducing intra-stream HoL blocking. Moreover, HBES outperforms SA-ECF in some cases where the FCT is longer due to harsh network conditions.

Then, we focus on how different schedulers handle high-priority streams of different sizes. For 10 KB and 50 KB streams, HBES significantly reduces their FCT by mitigating the inter-stream HoL blocking as shown in Fig. 13b and Fig. 13c, respectively. Though SA-ECF reacts to the stream priority, high-priority streams may sometimes be blocked when they get a backward transmission order. In comparison, HBES solves such an issue with SWRR algorithm. As for the 250 KB stream, the difference among schedulers is less significant because the network conditions impact the FCT of larger streams more, and inter-stream HoL blocking affects them less. However, HBES and SA-ECF can still reduce the FCT since they allocate more bandwidth to the high-priority stream.

Finally, by combining the results of the above experiments, we can conclude that, in a wide range of scenarios that users may encounter in mobile networks, HBES successfully achieves shorter FCT and less buffer occupation by mitigating HoL blocking. Additionally, evaluation of the two typical workflows also shows HBES is more capable of handling complex HTTP streams with priority information and speeding up web page object loading.

VI. RELATED WORK

Multipath transport protocols enable concurrent transmission over multiple network interfaces, which bring advantages such as bandwidth aggregation and connection robustness, but also pose challenges that single-path protocols never had [33], [37], [38], [39]. One known challenge is handling concurrent usage of multiple heterogeneous network paths with ever-changing conditions in wireless networks. Sending data through paths with different latency may cause out-of-order delivery, causing HoL blocking and excessive buffer usage. Worse still, the asymmetry between LTE and WLAN networks can be dramatic in mobile networks, which increases the difficulty of using multiple network paths efficiently.

Since it is hard for some naive schedulers (e.g., LRF and RR) to handle complex networks, several scheduling algorithms have been proposed to mitigate the OFO delivery issue. For example, DAPS (Delay-Aware Packet Scheduler [21]), which is designed as an extension to CMT-SCTP, makes scheduling decisions according to the one-way-delay and capacities of paths in order to reduce HoL blocking over asymmetric links. However, it is shown to generate spurious retransmissions, so BLEST (Blocking Estimation [19]) is

designed to reduce buffer blocking in a better way. Specifically, BLEST is designed to skip a certain currently available path, 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 [22]), ECF (Earliest Completion First [20]) and STTF (Shortest Transfer Time First [19]) try to estimate the arrival time of each packet more precisely by taking the queuing delay in send buffer into consideration. These three scheduling algorithms are similar to SAPS algorithm in HBES, but they are designed specially for MPTCP and cannot be directly used in the context of MPQUIC because stream multiplexing is not considered. Although Wu et al. [39] presented their implementation of a learning-based scheduling algorithm in MPQUIC, the stream multiplexing feature is still not discussed.

As a promising multipath protocol, there have already been many scheduling algorithms designed for MPQUIC, such as SA-ECF [30], PStream [31] and FStream [34]. Different from the aforementioned schedulers, they are all stream-aware and designed to handle multiplexing streams. Among them, PStream and FStream are designed to achieve optimal bandwidth allocation among different streams to avoid inter-stream HoL blocking. However, they both make a stream-level scheduling decision before streams are sent, which is efficient in static network conditions but not quite suitable for ever-changing mobile networks. Besides, in the cases that new streams are initiated during transmission, their scheduling decisions may become sub-optimal. SA-ECF, instead, makes per-packet scheduling which is more applicable to real traffic. Since extra delay is caused by HoL blocking at the end of stream transmission, SA-ECF may avoid sending data to a path with longer RTT if the stream can finish sooner through a faster but currently unavailable path. SA-ECF provides a simple yet efficient way to reduce HoL blocking delay, but high receive buffer occupation during transmission remains an unsolved problem.

Although it seems to be a conventional choice to concurrently transmit multiplexing streams as all MPQUIC schedulers mentioned above take such a measure (i.e., using WRR algorithm), Wang et al. [40] proposed a different stream scheduling policy called exclusive delivery. The exclusive delivery policy aims at reducing the average stream completion time by transmitting streams one by one. Since exclusive delivery may cause low-priority streams to starve, further research is needed to determine the pros and cons of different stream scheduling policies.

Recently, to improve certain video applications' performance, Zheng et al. [41] proposed XLINK as a cross-layer multipath extension for QUIC, which is driven by user-perceived QoE and built on top of a client-server feedback mechanism. These feedbacks from clients help XLINK "re-injecting" unacknowledged packets (i.e., sending duplicated packets) of a stream into a fast network path. Thus, it is able to shorten the completion time of currently high-priority streams and avoid intra-stream HoL blocking. XLINK achieved remarkable improvement in large-scale evaluation, but how it works with unchanged QUIC clients is not discussed.

Since we are trying to improve common HTTP traffic in mobile networks, our scheduler should always be ready to handle newly initiated streams of various priority and ever-changing network conditions, and we should not assume that clients run specific well-designed applications. As a result, in this paper, we choose SA-ECF as one of the comparison schemes, which is also a per-packet scheduling algorithm. Besides, HBES and SA-ECF need no prior knowledge, such as path latency and capacity, which some MPQUIC schedulers need to make stream-level scheduling decisions in advance. Overall, our contribution is to design a new per-packet scheduler for MPQUIC, which is able to handle complex HTTP-like multiplexed traffic in heterogeneous mobile networks.

VII. Conclusion

In this paper, we designed a new scheduler, HBES, to handle prioritized streams and make per-packet scheduling decisions to mitigate HoL blocking and excessive receive occupation caused by asymmetric and dynamic paths in mobile networks. Unlike existing MPQUIC schedulers, HBES uses a modified weighted round-robin algorithm to avoid inter-stream blocking caused by long scheduling cycles, ensuring that stream priority can be correctly reflected from the completion time. Then, HBES schedules each packet according to its arrival time by constantly tracking network conditions. Thus, it needs no prior knowledge about the network paths and makes real-time per-packet scheduling instead of assigning streams to certain paths in advance. That means that HBES is able to adapt to current network conditions and thus more applicable in changing mobile networks. We further implement HBES in user space based on MPQUIC open-source code. Moreover, our evaluation shows that HBES successfully achieves its goals to mitigate blocking issues as well as reduce buffer occupation, and when carrying traffic with typical patterns commonly seen in practice, HBES outperforms other schedulers in various simulated mobile network scenarios.

References

[1] K. Eaton. (Mar. 2012). *How One Second Could Cost Amazon $1.6 Billion in Sales*. Accessed: Oct. 2022. [Online]. Available: https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-salesl

[2] E. Schurman and J. Brutlag. (Jun. 2009). *Performance Related Changes and Their User Impact*. Accessed: Oct. 2022. [Online]. Available: https://www.youtube.com/watch?v=bQSE51-gr2s

[3] M. Belshe, R. Peon, and M. Thomson. (2015). *Hypertext Transfer Protocol Version 2 (HTTP/2)*. Accessed: Oct. 2022. [Online]. Available: https://rfc-editor.org/rfc/rfc7540.txt RFC 7540, IETF.

[4] P. Meenan. (May 2019). *Better HTTP/2 Prioritization for a Faster Web*. Accessed: Oct. 2022. [Online]. Available: https://blog.cloudflare.com/better-http-2-prioritization-for-a-faster-web/

[5] J. Iyengar and M. Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. Accessed: Oct. 2022. [Online]. Available: https://rfc-editor.org/rfc/rfc9000.txt

[6] A. Langley et al., "The QUIC transport protocol: Design and internet-scale deployment," in *Proc. Conf. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2017, pp. 183–196.

[7] (Sep. 2022). *HTTP/3 Protocol*. Accessed: Oct. 2022. [Online]. Available: https://caniuse.com/http3

[8] *Advances in Networking, Part 1 and Part 2*. Accessed: Oct. 2022. [Online]. Available: https://developer.apple.com/videos/wwdc2017/

[9] O. Bonaventure. (2019). *Apple Music on IOS13 Uses Multipath TCP Through Load-Balancers*. Accessed: Oct. 2022. [Online]. Available: http://blog.multipath-tcp.org/blog/html/2019/10/27/apple_music_on_ios13_uses_multipath_tcp_through_load_balancers.html

[10] Q. De Coninck and O. Bonaventure, "Multipath QUIC: Design and evaluation," in *Proc. 13th Int. Conf. Emerg. Netw. Experiments Technol.*, Nov. 2017, pp. 160–166.
[11] Q. De Coninck et al., "Pluginizing QUIC," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2019, pp. 59–74.
[12] Q. De Coninck and O. Bonaventure, "Multiflow QUIC: A generic multipath transport protocol," *IEEE Commun. Mag.*, vol. 59, no. 5, pp. 108–113, May 2021.
[13] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, and C. Paasch. (2020). *TCP Extensions for Multipath Operation With Multiple Addresses*. RFC 8684, IETF. Accessed: Oct. 2022. [Online]. Available: https://www.ietf.org/rfc/rfc8684.txt
[14] 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.
[15] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1529–1541, Jul. 2021.
[16] J. Liu and N. Kato, "A Markovian analysis for explicit probabilistic stopping-based information propagation in postdisaster ad hoc mobile networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 1, pp. 81–90, Jan. 2016.
[17] Y. Qiao, Y. Cheng, J. Yang, J. Liu, and N. Kato, "A mobility analytical framework for big mobile data in densely populated area," *IEEE Trans. Veh. Technol.*, vol. 66, no. 2, pp. 1443–1455, Feb. 2017.
[18] 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.
[19] 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.
[20] 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. EXperiments Technol. (CoNEXT)*, 2017, pp. 147–159.
[21] 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.
[22] 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.
[23] J. Han et al., "Leveraging coupled BBR and adaptive packet scheduling to boost MPTCP," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7555–7567, 2021.
[24] Y. Xing et al., "A low-latency MPTCP scheduler for live video streaming in mobile networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 11, pp. 7230–7242, 2021.
[25] Q. D. Coninck. (Dec. 2019). *Multipath QUIC*. Website of the Multipath QUIC Project. Accessed: Oct. 2022. [Online]. Available: https://multipath-quic.org/
[26] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation," in *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2012, pp. 253–264.
[27] (Oct. 2022). W3Techs. *Usage Statistics of HTTP/2 for Websites*. Accessed: Oct. 2022. [Online]. Available: https://w3techs.com/technologies/details/ce-http2
[28] M. Bishop. (2021). *Hypertext Transfer Protocol Version 3 (HTTP/3)*. Internet Engineering Task Force. Accessed: Oct. 2022. [Online]. Available: https://tools.ietf.org/html/draft-ietf-quic-http-34
[29] M. Wijnants, R. Marx, P. Quax, and W. Lamotte, "HTTP/2 prioritization and its impact on web performance," in *Proc. World Wide Web Conf. World Wide Web (WWW)*, 2018, pp. 1755–1764.
[30] 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.
[31] 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.
[32] 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.
[33] 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.
[34] X. Shi, L. Wang, F. Zhang, and Z. Liu, "FStream: Flexible stream scheduling and prioritizing in multipath-QUIC," in *Proc. IEEE 25th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2019, pp. 921–924.
[35] S. O'Dea. (2021). *4G and 3G Network Latency in the United States 2019*. Accessed: Oct. 2022. [Online]. Available: https://www.statista.com/statistics/818205/4g-and-3g-network-latency-in-the-united-states-2017-by-provider/
[36] C. Paasch, R. Khalili, and O. Bonaventure, "On the benefits of applying experimental design to improve multipath TCP," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol.*, Dec. 2013, pp. 393–398.
[37] C. Paasch and O. Bonaventure, "Multipath TCP," *Commun. ACM*, vol. 57, no. 4, pp. 51–57, Apr. 2014.
[38] 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.
[39] 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.
[40] J. Wang, Y. Gao, and C. Xu, "A multipath QUIC scheduler for mobile HTTP/2," in *Proc. 3rd Asia–Pacific Workshop Netw. (APNet)*, 2019, pp. 43–49.
[41] Z. Zheng et al., "XLINK: QoE-driven multi-path QUIC transport in large-scale video services," in *Proc. ACM Special Interest Group Data Commun. (SIGCOMM)*, 2021, pp. 418–432.

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

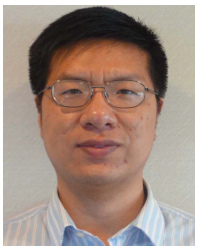
Kaiping Xue (Senior Member, IEEE) received the bachelor's degree from the Department of Information Security, University of Science and Technology of China (USTC), in 2003, and the Ph.D. degree from the Department of Electronic Engineering and Information Science (EEIS), USTC, in 2007. From May 2012 to May 2013, he was a Post-Doctoral Researcher with the Department of Electrical and Computer Engineering, University of Florida. Currently, he is 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 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 a (Lead) Guest Editor for many reputed journals/magazines, including IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS (JSAC), *IEEE Communications Magazine*, and *IEEE Network*. He is a fellow of the IET.

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

Jiangping Han (Member, IEEE) received the B.S. and Ph.D. degrees from the Department of Electronic Engineering and Information Science (EEIS), USTC, in July 2016 and 2021, respectively. She was a Visiting Student at Arizona State University in 2020 and 2021. She is currently a Post-Doctoral Fellow with the School of Cyber Science and Technology, USTC. Her research interests include data center network, 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 Fellow with the School of Cyber Science and Technology, USTC. His research interests include wireless communications, satellite-terrestrial integrated 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 a Faculty Member of Computer Science and Engineering at the University of Aizu, Japan (as an Associate Professor and then a Professor). He has authored or coauthored more than 130 technical papers in various archival journals and conference proceedings. He is currently a Professor with the Computer and Information Science Department, Fordham University. He has authored or coauthored more than 140 technical papers in the areas of distributed and parallel processing, wireless networks and mobile computing, optical networks, peer-to-peer communications, cloud and edge computing, cybersecurity, and quantum computing and communications in various archival journals and conference proceedings. The broad impact of his research can be verified by the fact that his works appear in 35 different journals, including 17 IEEE sponsored top-tier journals, such as IEEE TRANSACTIONS ON COMPUTERS, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY and the works also appear in top conferences, such as CVPR, INFOCOM, ICC, GLOBECOM, and SIGCOMM. 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 IEEE Computer Society and IEEE Communications Society.



Ruidong Li (Senior Member, IEEE) received the bachelor's degree in engineering from Zhejiang University, China, in 2001, and the Doctorate of Engineering 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 and the National Institute of Information and Communications Technology (NICT). He is also 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). 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 the IEICE.

Qibin Sun (Fellow, IEEE) received the Ph.D. degree from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 1997. Currently, he is a Professor with the School of Cyber Security, USTC. He has published more than 120 papers in international journals and conferences. His research interests include multimedia security and network intelligence and security.

Jun Lu received the bachelor's degree from Southeast University in 1985 and the master's degree from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 1988. Currently, he is a Professor with the Department of EEIS, USTC. His research interests include theoretical research and system development in the field of integrated electronic information systems. He is an Academician of the Chinese Academy of Engineering (CAE).