Multipath QUIC Implementation and Scheduling with Adversarial Multi-Armed Bandits

by

Shengjie Shu
Bachelor of Science (Honours), University of Victoria, 2021

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

Multipath QUIC Implementation and Scheduling with Adversarial Multi-Armed Bandits

by

Shengjie Shu
Bachelor of Science (Honours), University of Victoria, 2021

Supervisory Committee

Dr. Jianping Pan, Supervisor
(Department of Computer Science)

Dr. Yvonne Coady, Departmental Member
(Department of Computer Science)

Supervisory Committee

Dr. Jianping Pan, Supervisor
(Department of Computer Science)

Dr. Yvonne Coady, Departmental Member
(Department of Computer Science)

ABSTRACT

With the rapid development of computer networking, there has been an increase in the number of end devices with various types of network interfaces. Multipath transport-layer protocols have emerged as a potential solution to utilize multiple access paths and improve transmission capacity and reliability. One such protocol is *Multipath QUIC* (MPQUIC), which is a multipath extension to the QUIC protocol. Given the benefits of QUIC, MPQUIC is expected to be more promising in meeting the demands of future applications. This thesis explores the design and scheduling aspects of MPQUIC. We first study the QUIC protocol and propose a multipath implementation over it, called *a multipath extension to the QUIC module for ns-3*. Our implementation meets the demands for scalable multiple paths, flexible path schedulers, and compatible congestion control algorithms. Then, we further explore the scheduling problem in MPQUIC and design an intelligent multipath scheduling algorithm, called *multipath scheduling with adversarial multi-armed bandits*. A novel learning-based multipath scheduler, Multipath Scheduler with Adversarial Multi-Armed Bandit (MSAB), is presented, which makes an intelligent access network selection and is conscious of network heterogeneity and dynamics. Extensive simulation results with ns-3 show that the proposed MSAB scheduler can outperform state-of-the-art solutions 70% of the time with up to 30% higher goodput in highly dynamic scenarios.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

Dr. Jianping Pan, for providing me with countless opportunities during my Master's program. His guidance, support, encouragement, and patience have been invaluable to me throughout this journey. I am extremely grateful for the time he dedicated to mentoring me, teaching me how to conduct research, schedule my workload effectively, and interact professionally with others. I respect him deeply for his expertise and wisdom which have been instrumental in shaping my skills and capabilities. I want to acknowledge his continuous efforts and support in pushing me beyond my limits and helping me to achieve my academic goals. I express my sincere appreciation and admiration for his unwavering support and guidance throughout my Master's program.

My Parents, for always being supportive whenever it is needed. Their financial and emotional support have enabled me to pursue my academic dreams without any limitations.

I would also like to thank Wenjun Yang and Zhiming Huang for their help and support in this thesis, and all of my research group members for their assistance and inspiration throughout my master's program.

DEDICATION

To my lovely parents, Bin Shu and Shan Chen.

Chapter 1

Introduction

The development of computer networking has increased in recent years. Today's end devices contain many different types of network interfaces that function in various ways. For example, smartphones can connect to both WiFi and cellular networks. Similarly, a data center network can support several Ethernet connections [25]. For quick and reliable data sharing, multipath transport-layer protocols allow transmission with multiple paths in the transport layer, which enables the simultaneous use of various network access technologies. Specifically, the sender distributes application data across multiple available interfaces. The receiver reassembles and reorders data from various paths, making it transparent to the application. By doing so, multipath transport-layer protocols attempt to outperform single-path alternatives in terms of transmission capacity and reliability.

Multipath TCP (MPTCP) is one of the most well-known multipath transport-layer protocols [29]. Given that it was designed upon TCP, it had the opportunity to be implemented in some applications for commercial use. An example is its usage in Siri since iOS11 [7]. However, protocols that are constructed on top of the TCP/IP stack face a number of difficulties with the development of next-generation networks. Multipath QUIC (MPQUIC) [10] would be more promising to satisfy the demands of future applications as it is built upon QUIC which is a recently proposed transport-layer protocol to overcome the known issues. While the advanced features of QUIC, e.g., stream multiplexing and 0-RTT handshake, make the transmission more efficient, they also bring challenges for the design and implementation of the MPQUIC. For example, stream multiplexing creates difficulties in the acknowledgment of different paths. Thus, designing and implementing a functional MPQUIC is chosen as the starting point of this thesis.

On the other hand, the design of the multipath scheduling algorithm is one of the key components in multipath transport-layer protocols since it has a direct impact on how packets are distributed along each path, which in turn has an impact on transmission efficiency. The scheduler particularly decides how to distribute data over the access paths. It assigns each of the application's data packets to different interfaces in accordance with the scheduling strategy. The strategy of a multipath scheduler needs to consider many properties of transmissions, such as path throughput and delay, which is one of the most challenging components of the design process. Sent packets have a high likelihood of arriving at the destination out of order when the characteristics of different paths are diverse, particularly in terms of path latency and packet loss. This leads to some common transmission problems, such as connection breakage and Head-of-Line (HoL) blocking [32] issues, which lowers the aggregated performance. Existing multipath schedulers are either designed with some estimation-based models or dependent on strong assumptions, which might not be capable of different network scenarios. Therefore, we attempt to treat multipath scheduling as a decision-making problem and design an intelligent scheduling strategy with the goal of mitigating the aforementioned issues and enhancing overall performance.

Reinforcement Learning (RL) is a well-known method for solving decision-making problems. An RL agent constantly engages with its environment in an effort to learn the best course of action for maximizing cumulative reward. However, most of the traditional RL algorithm requires a heavy training process to get an optimal solution, which is not ideal when dealing with highly dynamic transmission problems. To this end, we construct the multipath scheduling problem as an online problem and formulate it with a lightweight RL method: Multi-armed Bandit (MAB). In a standard MAB problem, the objective of the gambler is to play an arm sequentially in every round and accumulate as many rewards as possible within a finite time horizon [22]. We refer to it as the stochastic MAB problem if the reward is chosen at random from a probability distribution. If not, we refer to it as the adversarial MAB problem. The adversarial MAB model is used in this thesis as it does not require the distribution assumptions of rewards.

In this thesis, we first study the QUIC protocol and implement a multipath extension over the QUIC module in ns-3 as the first work of this thesis. Since network transmission with multiple interfaces is desirable for the next-generation Internet to improve end-to-end performance and reliability, multipath QUIC (MPQUIC) is proposed to utilize multiple interfaces for Internet transmission with QUIC which

is already standardized and actively used in mainstream web browsers. However, the majority of the (MP)QUIC experimental platforms are built upon real systems or network emulators, which makes it challenging to investigate and experiment for further exploration. An MPQUIC simulation platform is still largely missing in the research community. As the first contribution of this thesis, we present an implementation of MPQUIC based on the QUIC module in ns-3, together with a description of the features that we implemented, verify the correctness of our implementation, and showcase the performance of MPQUIC with a set of experimentations. In our implementation, the scalability of multiple paths, the flexibility of path schedulers, and the extensibility of the congestion control algorithms are achieved.

In the second work of this thesis, we narrow down and focus on the multipath scheduling problem. State-of-the-art multipath schedulers are designed with some estimation-based models or learning-based policies, which depend on either strong assumptions or heuristic training approaches. Moreover, network heterogeneity and dynamic shifts are common in today's networks and are anticipated to become even more prevalent in the next generation of networking technologies. However, how to handle network heterogeneity and dynamics without relying on strong assumptions and heavy training processes remains an open issue. Therefore, we present a novel learning-based multipath scheduler, Multipath Scheduler with Adversarial Multi-Armed Bandit (MSAB), which makes an intelligent access network selection and is conscious of network heterogeneity and dynamics. Extensive simulation results with ns-3 show that the proposed MSAB scheduler can outperform state-of-the-art solutions 70% of the time with up to 30% higher goodput in highly dynamic scenarios.

1.1 Thesis Overview

Chapter 1 gives an introduction to this thesis, followed by an overview of the structure of the document itself.

Chapter 2 gives a detailed background of QUIC and MAB.

Chapter 3 presents the implementation of multipath QUIC in ns-3. This is the first of the two contributions expected in a thesis for a graduate degree. This work has been presented and published in WNS3 2023.

Chapter 4 presents the multipath scheduling with adversary multi-armed bandits. This is the second of the two contributions expected in a thesis for a graduate degree. This work has been submitted to GLOBECOM 2023.

Chapter 5 gives a conclusion of the thesis and presents future work.

Chapter 2

Background

2.1 Quick UDP Internet Connections (QUIC)

Quick UDP Internet Connections (QUIC), a general-purpose transport-layer network protocol, was developed by Google [21] in the beginning with the goal of improving HTTPS traffic performance. It also enables rapid deployment and continued evolution of transport mechanisms [21]. By using UDP, QUIC avoids some of the limitations and overhead associated with TCP, such as HoL blocking and the initial handshake round trips. It also incorporates features like encryption and congestion control.

QUIC offers several advantages over TCP, including reduced connection establishment time, faster data transfer, and improved congestion control. It achieves these benefits through features like stream multiplexing, which allows multiple independent streams of data to be sent over a single connection, and connection migration, which allows seamless transfer of connections between different network paths.

QUIC has gained significant attention and adoption in recent years. It has been integrated into popular web browsers like Google Chrome, Mozilla Firefox and Safari [15, 8, 2], and it is used by several large-scale online services. Additionally, the Internet Engineering Task Force (IETF) has been working on standardizing QUIC as an Internet protocol. The latest version, QUIC version 1 (QUICv1), is described in the RFC 9000 [17] as a proposed standard and a part of the HTTP/3 [4]. QUIC's design incorporates several practical and significant components. Only those features pertinent to and utilized in this thesis will be addressed further down.

Connection Establishment QUIC relies on a combined cryptographic and transport handshake for setting up a secure transport connection [21]. On a suc-

cessful 1-RTT handshake, a client caches information about the server. On subsequent connections to the same server, data can be delivered immediately after the client handshake packet without having to wait for a response from the server, which is known as a 0-RTT handshake. This allows the client to establish an encrypted connection with no additional round trips.

Figure 2.1 depicts a schematic of the handshake. Since the client initially does not know anything about the server, it must first send an inchoate client hello (CHLO) message to the server to elicit a reject (REJ) message before attempting a handshake. The REJ message contains the certifications and a connection ID. In subsequent handshakes, the client sends this token back to the server to show that the connection has been authenticated. The delivery of a complete CHLO with authenticated configurations follows. The connection ID is a unique integer identifier that serves routing and connection identification. The client caches the connection ID and uses it to initiate a complete CHLO when connecting to the same origin again. The client can now send initial-encrypted data to the server without having to wait for a response, as demonstrated in Figure 2.1, successful 0-RTT handshake.

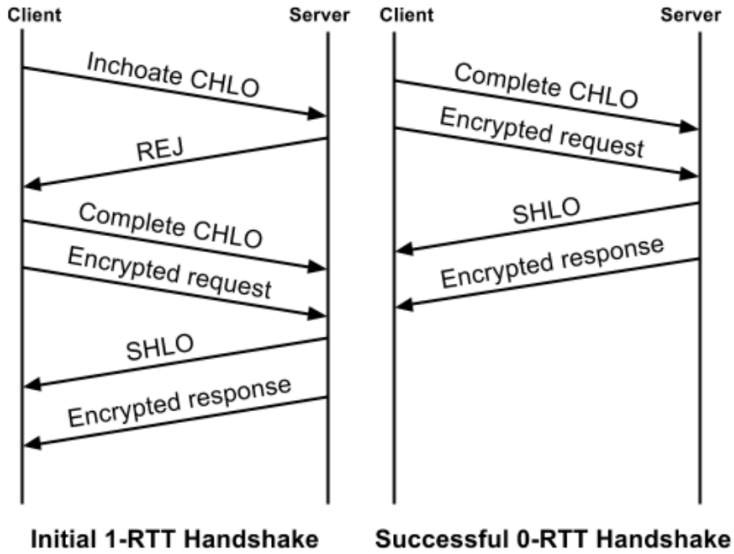


Figure 2.1: Timeline of QUIC's connection establishment [21].

Packet Header and Frame Structure With the exception of a few early handshake packets and reset packets, QUIC packets are fully authenticated and substantially encrypted. As depicted in Figure 2.2, a typical QUIC packet is composed of a common header followed by one or more frames. The parts of the QUIC packet header

outside the cover of encryption are required either for routing or for decrypting the packet: Flags, Connection ID, and Packet Number. The presence of the Connection ID field and the size of the Packet Number field are both encoded by flags. The Connection ID is utilized for identification and routing. It helps to locate the connection state and direct the connection's traffic to the right server. Both endpoints use the packet number as a per-packet nonce, which is placed outside of the encryption cover to support the decryption of packets received out of order, similar to DTLS [31].

The payload of QUIC packets, after removing packet protection, consists of a sequence of complete frames and may comprise multiple frames and multiple frame types [17]. Each frame begins with a Frame Type, indicating its type, followed by additional type-dependent fields. The transmission data is included in the Stream frame, the structure of which is depicted in Figure 2.2. To avoid head-of-line blocking due to TCP's sequential delivery, QUIC supports multiple streams within a connection, ensuring that a lost packet only impacts those streams whose data was carried in that packet. Subsequent data received on other streams can continue to be reassembled and delivered to the application. QUIC stream multiplexing is implemented by encapsulating stream data in one or more stream frames, and a single QUIC packet can carry stream frames from multiple streams [21].

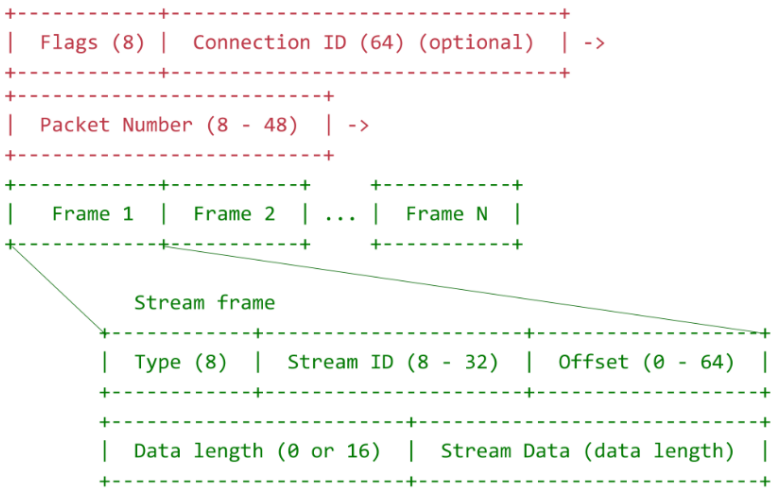


Figure 2.2: Structure of QUIC packet [21].

Loss Recovery and Error Control QUIC uses the packet number and the ACK frame to ensure reliability. Each QUIC packet carries a new packet number, even those with retransmitted data. This design solves the TCP retransmission ambi-

guity problem [19, 45] by obviating the need for a separate mechanism to distinguish the ACK of a retransmission from that of an original transmission. Stream offsets in Stream frames are used for delivery orders. The packet number represents an explicit time-ordering, which enables simpler and more accurate loss detection. QUIC acknowledgments explicitly encode the delay between the receipt of a packet and its acknowledgment being sent. Together with monotonically-increasing packet numbers, this allows for precise network round-trip time (RTT) estimation, which aids in loss detection. Figure 2.3 depicts the typical structure of an ACK frame.

```
ACK Frame {
  Largest Acknowledged (i),
  ACK Delay (i),
  ACK Range Count (i),
  First ACK Range (i),
  ACK Range (..) ...,
}
ACK Range {
  Gap (i),
  ACK Range Length (i),
}
```

Figure 2.3: Structure of QUIC ACK frame [21].

2.2 Multi-Armed Bandit (MAB)

The multi-armed bandit (MAB) problem is a problem in which a fixed limited set of resources must be allocated between alternative choices in a way that maximizes their expected gain when each choice's properties are only partially known at the time of allocation and may become better understood as time passes or by allocating resources to the choice [3]. This is a classic reinforcement learning problem that exemplifies the exploration-exploitation tradeoff dilemma. The name MAB was inspired by the idea of a gambler at a row of slot machines. Each machine provides a random reward from a probability distribution specific to that machine and the objective of the gambler is to maximize the sum of rewards earned through a sequence of lever pulls [36]. The crucial tradeoff the gambler faces at each trial is between "exploitation" of the machine that has the highest expected payoff and "exploration" to get more information about the expected payoffs of the other machines. In practice, MAB

has been used to model many decision-making problems such as the path scheduling problem in Chapter 4.

In a general MAB problem, we consider the basic model with independent and identically distributed (IID) rewards, called *stochastic bandits*. An algorithm has K possible actions or arms to choose from, and there are T rounds, for some known K and T. The algorithm chooses an arm and gathers a reward for this arm in each round. The objective is to maximize its overall reward across the T rounds. We base our predictions on three essential assumptions:

- The algorithm observes only the reward for the selected action, and it does not observe rewards for other actions that could have been selected.
- For each action a, there is a reward distribution $\mathcal{D}_a$ over real numbers. Every time this action is chosen, the reward is sampled independently from this distribution. The algorithm is initially unaware of the reward distributions.
- Per-round rewards are bounded and restricted to the interval $[0, 1]$ for the purpose of simplicity.

Algorithm 1 presents a basic abstract of the stochastic bandit.

Algorithm 1 Stochastic Bandit

```
for rounds t = 1, 2, ..., T do
    Learner selects exploitation or exploration advised by the algorithm.
    if exploitation selected then
        Learner selects the best action observed so far.
    else if exploration selected then
        Learner selects the action advised by the algorithm.
    end if
    Learner observes reward generated by D_a.
end for
```

Another variant of the multi-armed bandit problem is *adversarial bandit*. In this variant, an agent chooses an arm and an adversary simultaneously chooses the payoff structure for each arm at each round. Since it gets rid of all distributional presumptions, this is one of the strongest generalizations of the bandit problem [6]. Therefore, a solution to the adversarial bandit problem is a generalized solution to the more specific bandit problems. The adversarial bandit is utilized to address the path scheduling problem in the second work of this thesis.

Let $K > 1$ be the number of arms. A K-armed adversarial bandit is an arbitrary sequence of reward vectors $(x_t)_{t=1}^T$, where $x_t \in [0,1]^K$. In each round t, the learner chooses a distribution over the actions $P_t \in \mathcal{P}_{K-1}$. Then the action $A_t \in [K]$ is sampled from P_t, and the learner receives reward x_{tA_t}. The abstract of adversarial bandit is summarized in Algorithm 2.

Algorithm 2 Adversarial bandit

1: Adversary secretly chooses rewards $(x_t)_{t=1}^T$ with $x_t \in [0,1]^K$
2: **for** rounds $t = 1, 2, ..., T$ **do**
3: Learner selects distribution $P_t \in \mathcal{P}_{K-1}$ and samples A_t from P_t.
4: Learner observes reward $X_t = x_{tA_t}$.
5: **end for**

A policy $\pi : ([K] \times [0,1])^* \to \mathcal{P}_{K-1}$ is required in this setting to map the history sequences to distributions over actions [22]. There are a number of policies corresponding to different environments. Only the policies pertaining to our work are introduced in Sec. 2.2.1.

2.2.1 Exp3

Algorithm 3 Exp3

Input: K, T, η
1: Set $\hat{S}_{0i} = 0$ for all i
2: **for** rounds $t = 1, ..., T$ **do**
3: Calculate the sampling distribution P_t:

$$P_{ti} = \frac{\exp\left(\eta \hat{S}_{t-1,i}\right)}{\sum_{j=1}^{k} \exp\left(\eta \hat{S}_{t-1,j}\right)}$$

4: Sample $A_t \sim P_t$ and observe reward X_t
5: Calculate $\hat{S}_{ti}$:

$$\hat{S}_{ti} = \hat{S}_{t-1,i} + 1 - \frac{\mathbb{I}\{A_t = i\}(1 - X_t)}{P_{ti}}$$

6: **end for**

Exponential-weight algorithm for exploration and exploitation (Exp3), is the most basic method for adversarial bandits presented in Algorithm 3. Let $\hat{S}_{ti} = \sum_{s=1}^{t} \hat{X}_{si}$

be the total estimated reward by the end of round t, where $\hat{X}_{si}$ is the importance-weighted estimator of reward.

$$\hat{X}_{ti} = 1 - \frac{\mathbb{I}\{A_t = i\}}{P_{ti}}(1 - X_t). \tag{2.1}$$

It seems natural to play actions with larger estimated rewards with higher probability. While there are many ways to map $\hat{S}_{ti}$ into probabilities, a simple and popular choice is called exponential weighting, which for tuning parameter $\eta > 0$ sets

$$P_{ti} = \frac{\exp\left(\eta \hat{S}_{t-1,i}\right)}{\sum_{j=1}^{k} \exp\left(\eta \hat{S}_{t-1,j}\right)}. \tag{2.2}$$

The parameter η is called the learning rate. When the learning rate is large, P_t concentrates on the arm with the largest estimated reward and the resulting algorithm exploits aggressively. For small learning rates, P_t is more uniform, and the algorithm explores more frequently. Note that as P_t concentrates, the variance of the importance-weighted estimators for poorly performing arms increases dramatically. There are many ways to tune the learning rate, including allowing it to vary with time, which is used in the second work of this thesis and explained in Chapter 4.

Chapter 3

A Multipath Extension to the QUIC Module for ns-3

3.1 Introduction

Innovations in computer networking have grown rapidly over the past few decades. Today's end devices are equipped with multiple network interfaces in various modes. For instance, mobile devices connect to the Internet through both WiFi and cellular networks. Laptops similarly provide both Ethernet and WiFi adapters. Although single-path transmission still dominates the Internet because of the high cost of mobile data, or the limitations imposed by operating systems or network protocols, more and more researchers are beginning to concentrate on the use of multiple interfaces to improve network performance.

MPTCP [29] is a multipath extension built on top of TCP that allows for transmission with multiple paths in the transport layer. Thanks to its outstanding features such as throughput aggregation and congestion shift, it has been considerably adopted for commercial use. An example is the use of it since iOS11 for Siri [7]. Nevertheless, the next-generation networks pose a set of challenges to the protocols built upon the TCP/IP stack, e.g., connection breakage [30], and Head-of-Line (HoL) blocking issue [32]. To address the problems with TCP, Google first proposed Quick UDP Internet Connections (QUIC) [21], which was later standardized by the Internet Engineering Task Force (IETF) [17] as "QUICv1" in 2021. The innovative features of QUIC, such as stream multiplexing, frame structure, and 0-RTT handshake, enable it to improve transmission performance and easily adapt to various applications. The transmission

through QUIC or HTTP/3 is currently supported by an increasing number of web browsers and servers.

With the increasing popularity of QUIC, some limitations (e.g., lack of multipath management policies [33]) of QUIC were exposed. The multipath extension over QUIC (MPQUIC) also becomes an interesting topic for the research community. Motivated by the success of MPTCP [29], MPQUIC [10] is more promising to satisfy the demands of future applications. Even though MPQUIC is still under discussion by IETF, there are already some related works around the protocol designs [11], protocol implementations [35], scheduling strategies [38], and congestion control algorithms [43]. However, current experimental platforms for MPQUIC are mostly built on real systems or network emulators, which is challenging for the research community to investigate the potential of MPQUIC in diverse circumstances. A simulation platform of MPQUIC is to be developed and no widely available version exists yet. Therefore, we have developed an MPQUIC platform[1] based on the QUIC module [9, 28] for ns-3 since 2021. Our implementation and improvement achieve certain features, such as the scalability of multiple paths, the flexibility of switching multipath schedulers, and compatibility with different congestion control algorithms. In 2022, we fixed several bugs in the initial release and further enriched the set of path schedulers. Our implementation has been utilized in some research works [43, 42] since the first release.

In the remainder of this chapter, we first outline the key components of the MPQUIC protocol as well as the difficulties encountered during the code development in Sec. 3.2. Then, we detail the implementation of MPQUIC over the QUIC module for ns-3 in Sec. 3.3, including the information on the code structure, headers and frames, path management, path scheduler, and transmission reliability. In Sec. 3.4 we expand the evaluation of the performance of our MPQUIC implementation. Finally, Sec. 3.5 concludes the chapter and discusses future work.

3.2 MPQUIC Protocol Description

The multipath QUIC protocol intends to compensate for the missing features in QUIC by utilizing different paths that exist between a client and a server [10]. The layered structure of MPQUIC is illustrated in Figure 3.1. Differing from QUIC which

[1] `https://github.com/ssjShirley/mpquic-ns3`

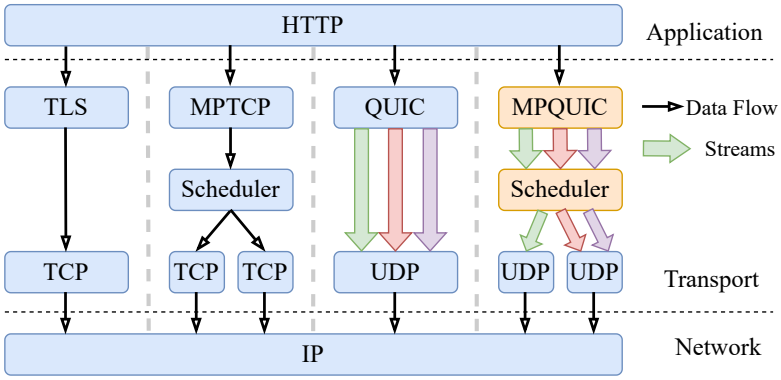


Figure 3.1: Structure of MPQUIC in comparison with others.

delivers data directly onto UDP, an MPQUIC layer is placed between the application and QUIC protocol to deal with the multipath connection, following the MPTCP [29] design logic. MPQUIC inherits the stream multiplexing feature in QUIC, allowing it to transmit data in several independent streams. The scheduler then arranges the data streams onto appropriate UDP paths for the multipath transmission.

3.2.1 MPQUIC

The motivation for extending multipath capability over QUIC protocol is to associate resources of distinct paths within a single connection and realize a smooth migration between different interfaces [26]. The single-path QUIC integrates the features of TCP, TLS, and a portion of HTTP/2 over the UDP transmission [21]. QUIC works differently than traditional TCP connections in that it mitigates the head-of-line blocking issue, shortens the transmission latency, and incorporates encryption. Based on several salient features and improvements in QUIC, the design specifications of MPQUIC [10, 35] are described in the following five components.

Path Identification. The increasing packet numbers are used to identify the lost packet in a single-path QUIC connection. If all packets share one numbering space in MPQUIC and are sent over different paths, they may arrive out of order, resulting in a misinterpretation of packet loss. To address this problem, MPQUIC includes a path identification in the packet header and creates a per-path numbering space, which isolates packet numbers on various paths from one another and restricts the sequential feature to that path. This could prevent middleboxes from accidentally dropping the packets that have the same packet number but in different paths.

Path Management. A path manager handles the path creation and removal in

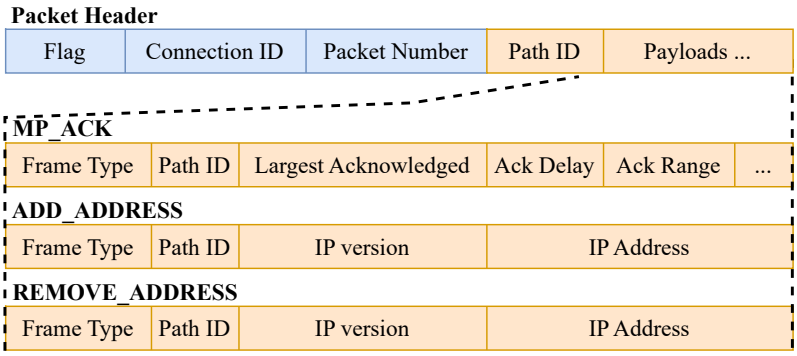


Figure 3.2: MPQUIC header and new frames.

MPQUIC. Figure 3.2 shows that the packet payload in QUIC is comprised of multiple frames which can store stream data or control information. Benefiting from this frame structure, MPQUIC can define some specific types of frames to store the multipath information. To manage multiple paths, new frames, e.g., `ADD_ADDRESS` and `REMOVE_ADDRESS`, are introduced to help the path establishment and removal. Furthermore, since the zero round-trip time (0-RTT) is supported by the Connection ID, MPQUIC is able to authenticate new paths within one handshake, while MPTCP requires a three-way handshake before being able to use any paths.

Reliable Data Transmission. `MP_ACK` frame is defined to handle the packet loss recovery in MPQUIC. Since the acknowledgment depends on the packet numbers and it needs to contain all unacknowledged packet numbers, multipath transmission with per-path numbering space would cause conflict and an oversized frame if we still use the original `ACK` frame. Therefore, `MP_ACK` frame is introduced to split the huge acknowledgment into smaller path-based frames to prevent conflict. Also, MPQUIC can transmit `MP_ACK` frame over different paths, in contrast to MPTCP which must return acknowledgment on the same path as the data received [29].

Packet Scheduling. The path scheduler in MPQUIC is responsible for allocating packets onto different paths. The simplest scheduling algorithm is Round-Robin (RR), which can schedule packets on different paths sequentially, but may cause high latency when two flows have a large difference in bandwidth and RTT. So, MPQUIC uses Minimum-RTT (MRTT) by default, which is implemented in the Linux kernel for MPTCP. Provided that the congestion window still has space, the path with the lowest measured RTT is preferred. In addition, several advanced scheduling algorithms have been proposed. For example, BLEST [12] and ECF [23] schedule packets by estimating how many packets could be sent on the fast path during the

RTT of the slow path. Peekaboo [38, 39] is an adaptive multipath scheduler based on reinforcement learning algorithms.

Congestion Control. The congestion control of transport-layer protocols is critical for smooth and efficient transmission by determining when and how to adjust the sending window. NewReno [40] and CUBIC [14] are widely used for single-path TCP and QUIC, but they will cause unfairness in multipath protocols [37]. Thus, the default congestion control algorithm in MPQUIC incorporates OLIA [20], which integrates the information from all paths and also presents a good performance in MPTCP.

3.2.2 Challenges of MPQUIC Implementation in ns-3

The code implementation of MPQUIC is based on the QUIC module in ns-3. Since this module is designed for single-path connections, it necessitates many adjustments when it comes to multipath connections while maintaining original transmission functions at the same time. Challenges also appear with the modifications.

First, only one path connection is considered in QUIC. The host address and peer address are singular for a single connection and directly assigned by the applications. As a result, the ns-3 QUIC module does not support operations with multiple pairs of addresses. However, the capability to explore multiple local addresses and advertise the addresses to peers is necessary for multipath transport-layer protocols. Therefore, such features should be incorporated into our implementation.

The other challenge is that the functions of sending and receiving in the QUIC module are interlocking and linked one by one. If multiple paths are just added to some of the essential functions, it would disrupt the entire transmission process. As a result, we have to thoroughly examine the data flow and implement multipath features in all relevant functions. Specifically, `m_tcb`, an instance of `QuicSocketState`, is used throughout the QUIC module, containing the loss detection variables as well as the congestion control management states, such as round-trip time, acknowledgment delay, loss event, etc. However, in a multipath implementation, each path should maintain an independent space to manage such control information. A similar situation also exists in `QuicSocketTxBuffer`, which is not only a simple storage list of frames but also responsible for frames splitting and reassembling based on the available window and maximum transmission units. Thus, our implementation cannot directly instantiate the buffer object for each path. Instead, an independent

space of `QuicTxPacketList`, storing the sent packets in the buffer, is required for each path. Furthermore, the congestion control algorithm design of multipath transmission should involve factors of different paths in a coupled way. Although the congestion control is extensible in QUIC, the interfaces of `QuicCongestionOps` only consider single-path factors, which is not sufficient. Also, a multipath-based scheduler is missing in the QUIC module. Therefore, we establish a new congestion control class that involves multipath factors and maintains extensibility, and a scheduler class that is flexible and compatible with various algorithms.

Overall, the major challenges for MPQUIC implementation are address advertisement, path separation, and algorithm extension based on the ns-3 QUIC module, maintaining original transmission functions at the same time. The detailed implementation is described in the next section.

3.3 Implementation of MPQUIC in ns-3

The key characteristics of the multipath QUIC implementation for ns-3 are described in this section. The UML diagram for MPQUIC is shown in Figure 3.3. As the diagram shows, the primary classes for MPQUIC are developed on top of the QUIC module [9] in ns-3. New classes, functions, and variables have been introduced to MPQUIC to account for the multipath features.

3.3.1 Code Structure

To implement MPQUIC, we need to add new components to store the multipath information, build a path manager to control multiple paths and implement schedulers to allocate the packets transmitting on each path and the congestion control algorithms collaborating with multipath states. As the UML diagram (Figure 3.3) shows, some new classes are generated for multipath realization, including `MpQuicSubflow`, `MpQuicPathManager`, `MpQuicScheduler`, and `MpQuicCongestionOps`, which store the path states, and perform the operations of path management, packet arrangement, and congestion control. Corresponding objects are instantiated in `QuicSocketBase` that handles the basic transmission functions in QUIC.

The original classes in the QUIC module are also enhanced with new components. The additional variables and functions defined in `QuicL4Protocol` and `QuicSocketBase` are responsible for associating the multipath features with the original

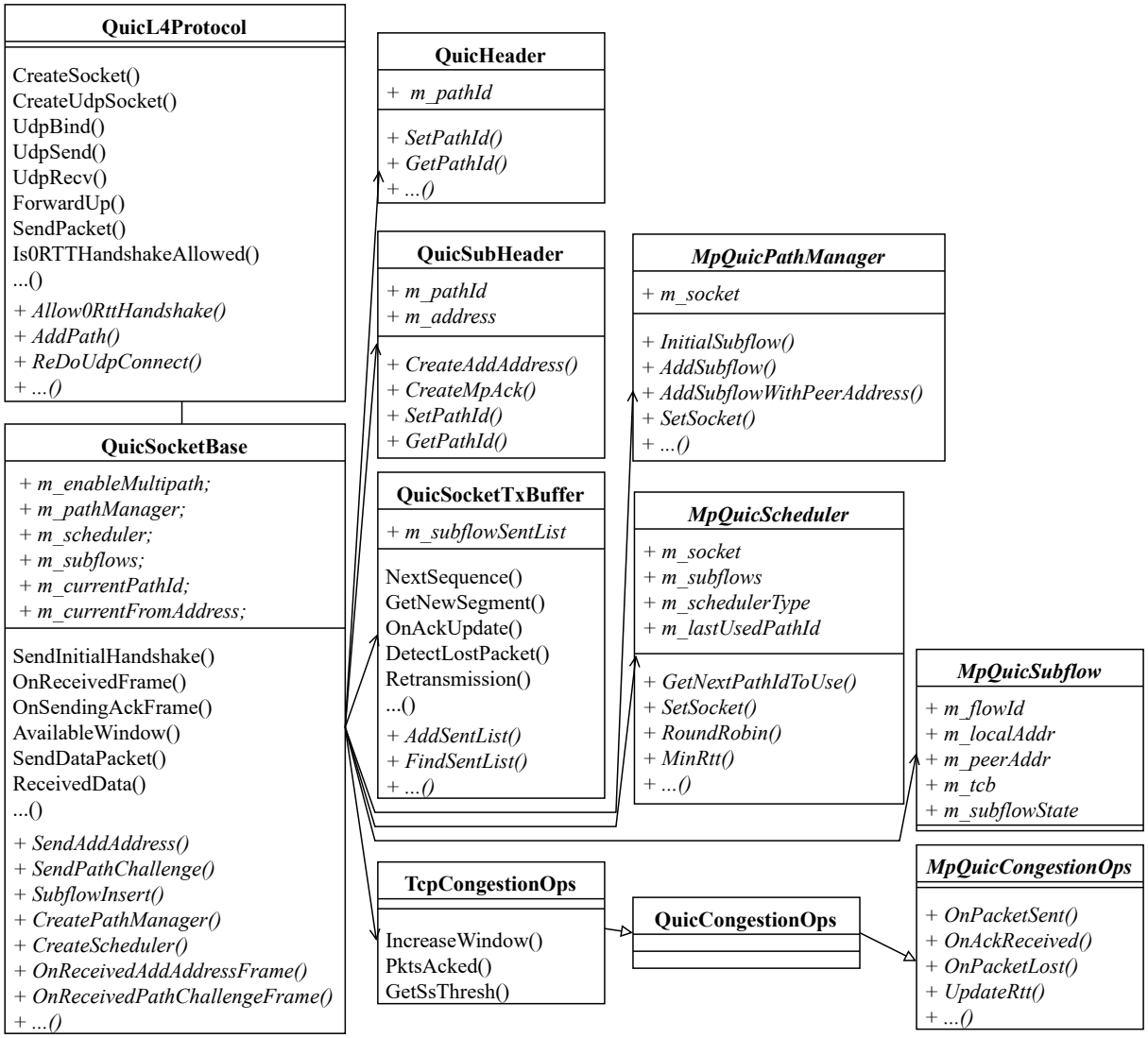


Figure 3.3: MPQUIC UML diagram (new classes, functions, and variables shown in italics).

QUIC functions and interacting with the related classes processing path establishment, packet arrangement, and congestion control. The transmission buffers for multiple paths are operated by the added variables and functions in `QuicSocketTxBuffer`. Furthermore, the additional components in `QuicSubheader` deal with the newly defined frames in MPQUIC. The newly defined header is handled by the added components in `QuicHeader`.

Overall, our design attempts to retain the transmission logic and utilize the existing classes in the original QUIC module as much as possible when building multipath

features above it.

3.3.2 MPQUIC Headers and Frames

In order to distinguish packets transmitted over different paths, a new variable `m_pathId` is defined in the `QuicHeader`. Figure 3.2 depicts the structure of the MPQUIC header and new frames. MPQUIC uses the `MP_ACK` frame to maintain reliable data transmission. A static function `CreateMpAck()` is added to `QuicSubheader` for the `MP_ACK` creation, which is invoked by `QuicSocketBase`. As the frame identifies the acknowledged packets with the packet number, per-path numbering space will create confusion when acknowledging packets with the same packet number. Thus, `m_pathId` is also declared in `QuicSubheader` to avoid confusion by distinguishing acknowledged packet numbers from different paths.

Moreover, multipath transmission necessitates the exchange of new addresses between two hosts. To meet this requirement, `ADD_ADDRESS` and `REMOVE_ADDRESS` are added to `QuicSubheader`, and `m_address` is defined as well as `CreateAddAddress()` and `CreateRemoveAddress()` accordingly. The function `OnReceivedAddAddressFrame()` and `OnReceivedRemoveAddressFrame()` are also defined to process the corresponding frames in `QuicSocketBase`. Although `PATH_CHALLENGE` and `PATH_RESPONSE` have already been included in `QuicSubheader` as mentioned in the IETF standard of QUIC [17], the existing QUIC module does not construct the functions to process these frames. As they are required to address migration and multipath implementation, we also define `OnReceivedPathChallengeFrame()` and `OnReceivedPathResponseFrame()` in `QuicSocketBase` to handle these two frames.

3.3.3 MPQUIC Path Management

The MPQUIC path creation and removal are handled by the `MpQuicPathManager`. The MPQUIC connection is established with a single-path connection first since the multipath transmission between two hosts relies on a stable connection. To distinguish it from the other paths established later, we name the first established connection as the main flow or subflow 0. Noting that the subflow is equivalent to an end-to-end path or connection, we will use them interchangeably throughout the thesis. In the rest of this section, we will use subflow with numbers to represent different paths and subflow 0 to describe the main flow. `InitialSubflow()` in `MpQuicPathManager` is used to instantiate `MpQuicSubflow` for the main flow, which is invoked in `QuicSocketBase`

during the handshake process of the connection establishment. It instantiates the flow ID, the local address, and other subflow components.

Figure 3.4 illustrates the procedures of the subflow establishment. After the handshake process in the main flow, the server begins to actively establish new subflows when the multipath feature is enabled (i.e., `m_enableMultipath=True`). `AddSubflow()` is invoked to explore the available interfaces and instantiate `MpQuicSubflow`. Then, the client establishes new subflows passively after receiving `ADD_ADDRESS` from the server side. `AddSubflowWithPeerAddress()` is invoked to instantiate `MpQuicSubflow` for the client side and establish the subflows with the advertised new addresses. New UDP sockets are also declared in `QuicL4Protocol` with `AddPath()` after receiving the new addresses. Benefiting from the Connection ID and 0-RTT handshake in QUIC, a new subflow establishment does not need both hosts to send `ADD_ADDRESS` for address authentication. By providing the authenticated Connection ID in the packet header, the server can authenticate the new client subflow without a three-way handshake.

The state machine of subflow maintenance is shown in Figure 3.5. Once the subflow is instantiated but not authenticated yet, its state is VALIDATING. After the authentication process, the subflow state becomes ACTIVE, implying that it is able to transmit data. The subflow removal process is triggered once a path is abandoned or the transmission is complete. The active host sends `PATH_ABANDON` and its state becomes CLOSING. The passive host then sends `REMOVE_ADDRESS` and its state changes to CLOSING. The active host state turns to CLOSED when it receives `REMOVE_ADDRESS`. Also, if the transmission times out due to an unstable connection, the state will turn to CLOSED and remove the related subflow.

3.3.4 MPQUIC Path Scheduler

The `MpQuicScheduler` class is designed for applying various scheduling strategies in MPQUIC. `m_schedulerType` is an attribute for the customization purpose, which implies that users configure it depending on different requirements. Currently, our implementation supports more scheduling algorithms, including RR, MRTT, BLEST, ECF, and Peekaboo. `GetNextPathIdToUse()` decides which subflow will be used to send out packets, which is invoked by `QuicSocketBase` in `SendDataPacket()`. Accordingly, `m_subflows` and `m_lastUsedPathId` are declared for the scheduler.

Since `MpQuicScheduler` is defined independently, the flexibility of utilizing differ-

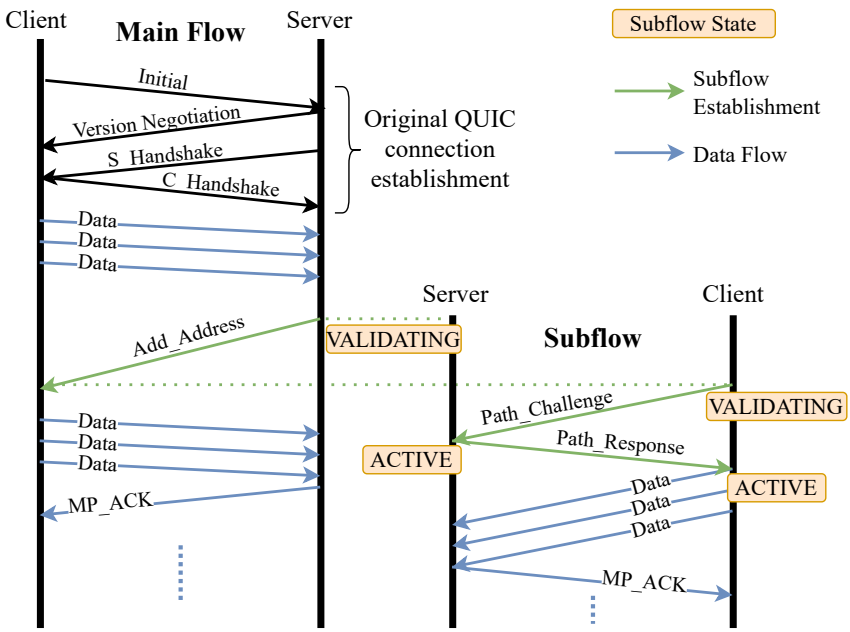


Figure 3.4: Procedures for subflow establishment.

ent scheduler algorithms is provided in our implementation, which keeps the extensibility for further research.

3.3.5 Data Flow in an MPQUIC Connection

The data transmission in QUIC is encapsulated with the stream frames, which correspond to `QuicL5Protocol` and `QuicStreamBase`. These two classes are independent of the multipath features so that the process of data encapsulation in MPQUIC is the same as what QUIC did [9]. The challenge for the data flow in an MPQUIC connection thus focuses on the packet sending and receiving process during the transmission.

For the packet-sending process, the raw data from the applications are appended to the variable `m_txbuffer` after being encapsulated into stream frames. `QuicSocketTxBuffer` then identifies whether the frame is for a data stream or a control frame. The control frames are stored in `m_streamZeroList` and data stream frames are stored in `m_subflowSentList` depending on their sending Path ID set by `MpQuicScheduler`. `m_subflowSentList` is a vector of `QuicTxPacketList` which is a linked list for transmitted packets in `QuicSocketTxBuffer`. In `QuicSocketBase`, `SendPendingData()` picks and sends out the packets from `QuicSocketTxBuffer` after checking the sending subflow's `AvailableWindow()`. Then, the sent packets are stored in `m_-`

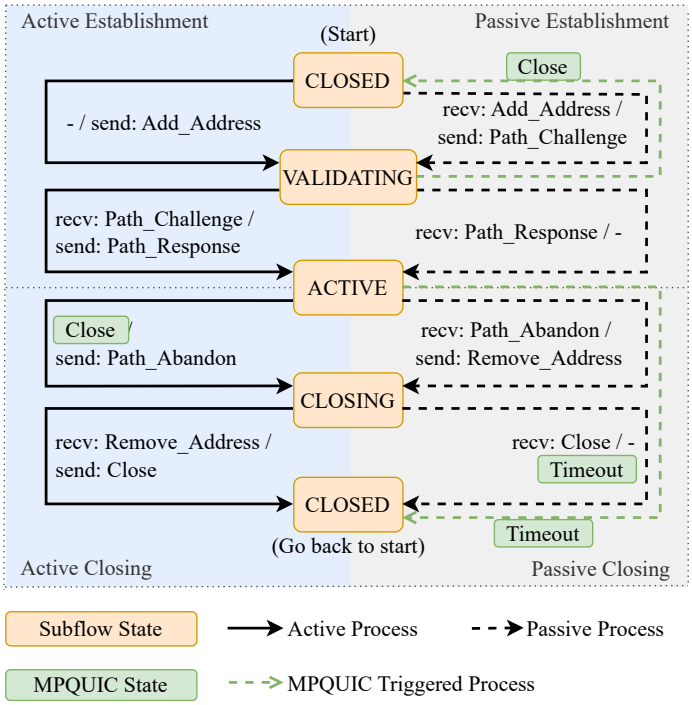


Figure 3.5: State machine of a subflow.

`subflowSentList` for the process of loss recovery and error control afterward.

For the packet receiving process, the callback functions `ForwardUp()` in `QuicL4Protocol` are triggered when receiving a packet from the UDP socket, which checks for the address authentication and forwards the packet to `ReceivedData()` in `QuicSocketBase`. Then, the packet is dispatched into frames to distinguish data streams from control frames. With the Path ID provided in the packet header, `MaybeQueueAck()` and `SendAck()` create and send out `MP_ACK` when receiving the data streams. `OnFrameReceived()` further deals with the received control frames, where the frame type is checked first and the corresponding functions are invoked.

3.3.6 Transmission Reliability

MPQUIC maintains the reliability of transmission at both the stream level and the packet level. The stream frame and offset are responsible for the stream-level error control, which has been implemented in the existing QUIC module. At the packet level, `MP_ACK` is transmitted to recognize the lost packets in MPQUIC. Recalling that the sent packets are stored in `m_subflowSentList`, `MpQuicTxBuffer` marks the lost packets in `m_subflowSentList` with the information provided from `MP_ACK`. `On-`

`ReceivedAckFrame()` then matches the lost packets on the corresponding subflows and does the retransmission with the function `DoRetransmit()` achieving the loss recovery for each subflow. QUIC also supports retransmission with timeout by `ReTxTimeout()`, which is still supported in MPQUIC and subflow-based. Each subflow has its own timeout alarm stored in `m_tcb`, which is an instance of `QuicSocketState` declared in `MpQuicSubflow`. If the timeout is triggered, it will retransmit on the same subflow.

3.3.7 Congestion Control

`MpQuicCongestionOps` is created to process the congestion control algorithms in MPQUIC. In the QUIC module, it supports pluggable congestion control with `QuicCongestionOps`, `TcpCongestionOps`, `TcpNewReno`, etc. We maintain this feature and plug it with the OLIA congestion control algorithm when multipath is enabled. Since OLIA is the default in MPQUIC for fairness in multipath protocols, we implement it in `MpQuicCongestionOps`, containing `OnPacketSent()`, `OnAckReceived()`, `OnPacketLost()`, and `UpdateRtt()`. These functions are invoked by `SendDataPacket()` and `OnReceivedMpAckFrame()` in `QuicSocketBase`. The congestion control is also extensible in MPQUIC by inheriting the `MpQuicCongestionOps` class.

3.3.8 Current Status

We started our implementation in 2021 and kept improving it along with the update of the IETF draft. Here are the supported features that our implementation aligns with the latest IETF draft [24]. Our implementation supports most of the features mentioned in [24], including handshake negotiation and transport parameters (Section 3.3.2), path setup and removal (Section 3.3.3), multipath operation with multiple packet number spaces (Section 3.3.6), path scheduling (Section 3.3.4), and congestion control (Section 3.3.7). The current version fixes several bugs in the initial release. For example, we realized that the packet size decreased down to a small number when transmitting with the RR scheduler due to the unexpected path close flag. Our implementation has been utilized in a few recent research works [43, 42] since then. We also enriched the set of path schedulers to further improve its flexibility.

Compared to the latest IETF draft of MPQUIC [24], our implementation omits some features that have little bearing on the basic transmission and we will implement them in our future work. [24] introduces a `PATH_STATUS` frame that informs the peer

to send packets in the preference. Our implementation is able to have the path status locally as Figure 3.5 shows, however, we do not support transmitting the path status between peers currently. In the path close stage, [24] includes a `RETIRE_CONNECTION_ID` frame to indicate to the receiving peer that the sender will not send any packets associated with the Connection ID used on that path anymore, which is not included in our current version. On the other hand, our implementation of MPQUIC currently only supports transmitting `MP_ACK` with the path given by the coming packets. An `MP_ACK` frame that can be returned via a different path is considered as our future work. Additionally, [24] indicates the packet protection with TLS and AEAD for connection ID, while the security features are not considered in our current stage. We will continue to improve our implementation to align with the latest IETF draft.

3.4 Evaluation

To demonstrate the behavior of the MPQUIC implementation in ns-3, a set of use cases are presented. We first show the scalability of our implementation. Then, we illustrate the performance with different congestion control algorithms and schedulers. Figure 3.6 presents the network topologies [16] used in our evaluation and Table 3.1 details the path settings. The example scripts we used to run the following experiments can be found in the `scratch` directory in our GitHub repository.

(a) Four paths (b) Two paths

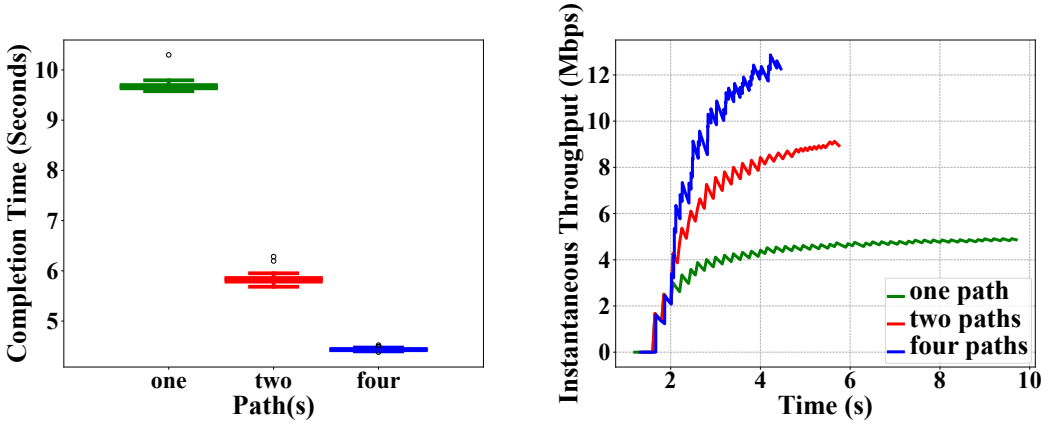
Figure 3.6: Topology with multiple paths.

3.4.1 Scalability of Multiple Paths

It is important to note that the number of paths available on end devices is typically limited. While it is technically possible to have a large number of paths, the practical limit is influenced by factors like hardware capabilities, software support, network infrastructure, and the intended use case. In most scenarios, end devices commonly

Table 3.1: Path Settings

Setting	Bandwidth	OWD	Loss
0	5–5.5 Mbps	50–55 ms	0–0.08%
1	10–11 Mbps	10–11 ms	0–0.08%
2	5–5.5 Mbps	10–11 ms	0–0.01%
3	10–11 Mbps	50–55 ms	0–0.01%

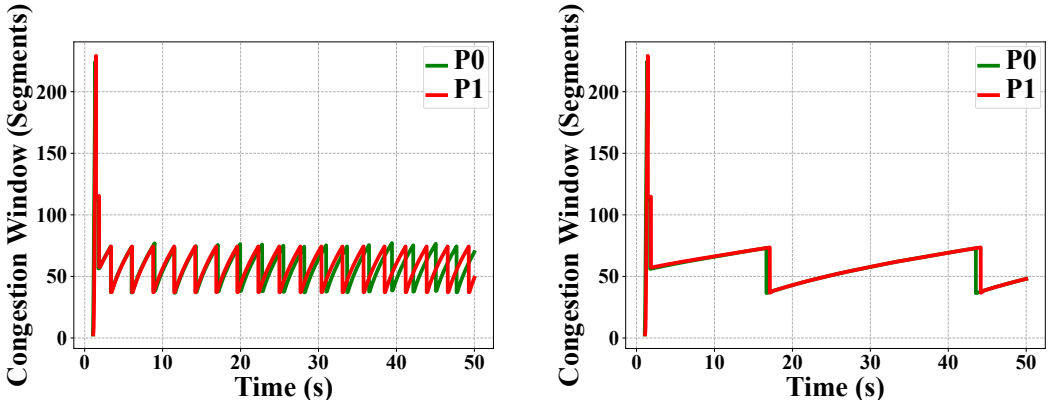


(a) Completion time

(b) Instantaneous throughput

Figure 3.7: Completion time and instantaneous throughput comparison for one, two, and four paths.

have two to three paths available for multipath connections. This includes combinations like Wi-Fi, cellular, or Ethernet. Thus, the scalability of our implementation is demonstrated with the topology of four paths between client and server (Figure 3.6a), in which each path has the same range of data rate and one-way delay (OWD) under Setting 0 as listed in Table 3.1. Figure 3.7a illustrates the completion time of transmitting 5 MB data with one, two, or four paths. Each of them runs 50 rounds and randomizes under a uniform distribution in terms of the path setting. The completion time of four paths is shorter than transmitting with one-path and two-path. Figure 3.7b depicts the instantaneous throughput for one, two, and four paths. The instantaneous throughput of four paths is much higher than the one-path and two-path scenarios. Overall, the performance is significantly improved with four paths compared with the single-path transmission, which presents the ability to simultaneously utilize multiple paths and improve the performance with our implementation.



(a) NewReno (b) OLIA

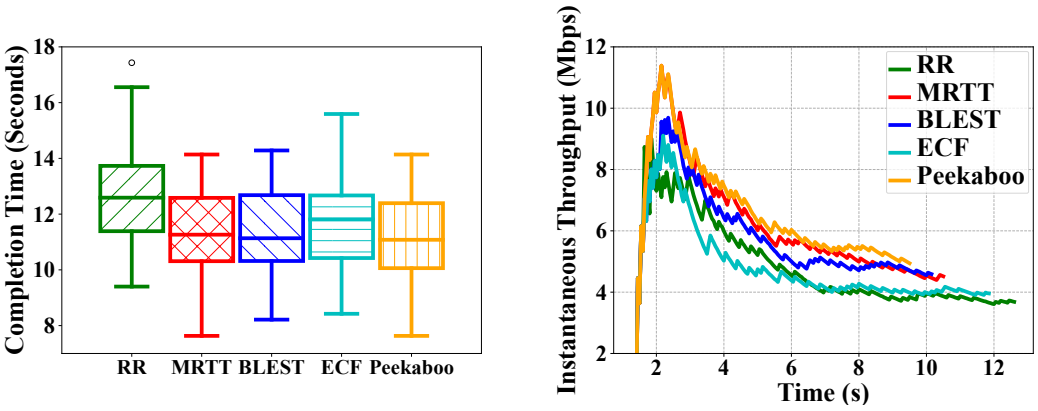
Figure 3.8: Congestion window comparison for NewReno and OLIA.

3.4.2 Performance of Different Congestion Control Algorithms

The evaluation of the congestion control algorithms uses the two-path topology (Figure 3.6b) and explores under Setting 1 on each path. RR is the default scheduler in this evaluation. Figure 3.8 presents the congestion window of NewReno and OLIA by transmitting unlimited data in 50 seconds, where the segment size is 1460 bytes. NewReno shows a more frequent change and the congestion windows of two paths stagger over time as NewReno is designed for single-path transmission and there is no interaction between two paths. The congestion windows with OLIA grow more general and the trade of the congestion window of two paths is much close since there are interactions among multiple designed by OLIA.

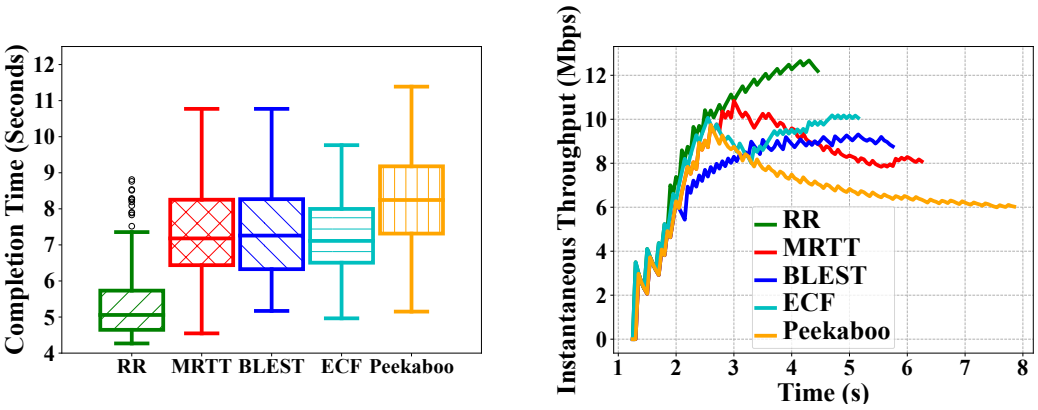
3.4.3 Flexibility of Different Path Schedulers

The flexibility of our implementation is tested with five different types of schedulers, using OLIA as the default congestion control algorithm. We consider both dominating and competing scenarios under the two-path topology (Figure 3.6b). In a dominating scenario, P0 uses Setting 0 and P1 uses Setting 1, indicating that P1 should always be faster than P0 with higher bandwidth and lower OWD. Figure 3.9a and 3.9b depict the completion time and the instantaneous throughput of transmitting 5 MB data with different schedulers for 50 rounds. RR has the longest completion time and lowest throughput in the dominating scenario as the performance of RR is restricted by the slow path. Schedulers like MRTT and Peekaboo have a better performance in



(a) Completion time (b) Instantaneous throughput

Figure 3.9: Completion time and instantaneous throughput comparison in the dominating scenario.



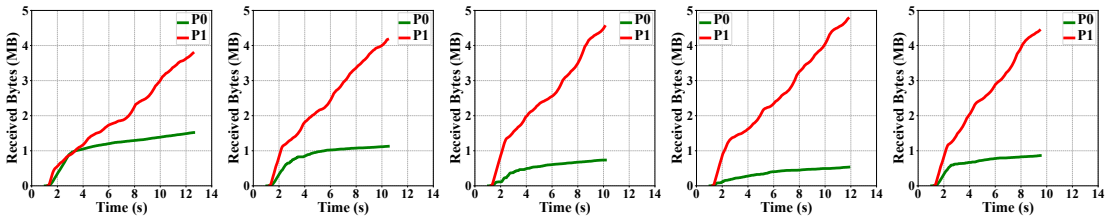
(a) Completion time (b) Instantaneous throughput

Figure 3.10: Completion time and instantaneous throughput comparison in the competing scenario.

this scenario since they transmit more packets on the fast path.

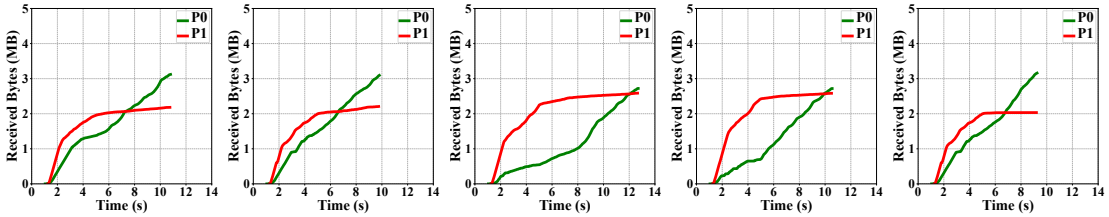
Figure 3.10a and 3.10b depict those performances in a competing scenario in that P0 uses Setting 2 and P1 uses Setting 3, indicating P0 has higher bandwidth and higher OWD while P1 has lower bandwidth and lower OWD. In the competing scenario, RR has the lowest completion time and the highest instantaneous throughput as it equally utilized each path while other schedulers always tend to transmit on a better path, ignoring the other competing path.

To present the detailed performance of each path, we also depict the received bytes of each path. Figure 3.11 presents the received bytes for different schedulers



(a) RR (b) MRTT (c) BLEST (d) ECF (e) Peekaboo

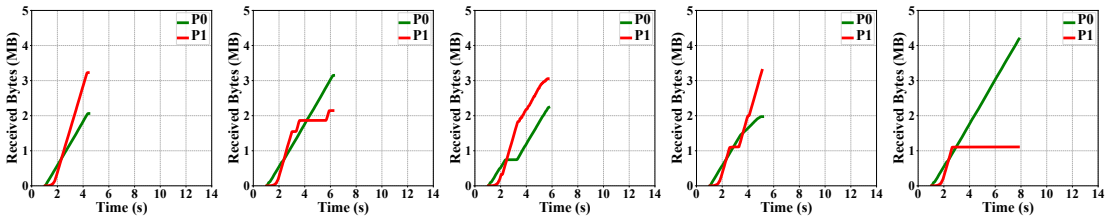
Figure 3.11: Received bytes of two paths in the dominating scenario.



(a) RR (b) MRTT (c) BLEST (d) ECF (e) Peekaboo

Figure 3.12: Received bytes of two paths in the dominating scenario with swapped setting after 5 seconds.

in the dominating scenario. Since P1 is faster than P0, P1 received more bytes with all schedulers. Figure 3.12 also presents the dominating scenario but two paths swapped their settings after 5 seconds. Specifically, at the start of the transmission, P0 uses Setting 0 and P1 uses Setting 1. After 5 seconds, P0 uses Setting 1 and P1 uses Setting 0. By doing so, we are able to explore the adaptivity of different schedulers. RR, MRTT, and Peekaboo are able to reflect the path change quickly as the received bytes are crossed around 6 seconds, while BLEST and ECF do the scheduling strategy by estimating from all previous records, which brings a worse performance in this scenario. Furthermore, Figure 3.13 explores the performance of different schedulers in the competing scenario. The received bytes of RR grow fast and smoothly benefiting from its equal use of each path, while the learning strategy of Peekaboo tends to select P0 when two paths are competing, which leads to the worse performance in this scenario. Overall, our implementation is able to utilize different schedulers, and their performance under different scenarios is expected.



(a) RR (b) MRTT (c) BLEST (d) ECF (e) Peekaboo

Figure 3.13: Received bytes of two paths in the competing scenario.

3.5 Summary

In this chapter, we presented an ns-3 implementation of MPQUIC. We overcame the challenges of advertising multiple addresses, separating transmission paths, and extending the scheduling and congestion control algorithms, while still maintaining the original transmission features. With a set of experimentations, we evaluated the correctness of our implementation, the scalability of multiple paths, the flexibility of the path schedulers, and the congestion control algorithms, which provide a stable simulation platform for the research community on multipath transport protocols. We will further implement the currently missing features in MPQUIC ns-3 and keep updating according to the IETF draft.

Additionally, the scheduler design in MPQUIC is crucial as it determines how data is distributed across multiple paths, exploiting their diverse characteristics for improved throughput. Path selection, load balancing, and effective network resource utilization are all impacted by the scheduling algorithm. While congestion control is important, the scheduler's role in leveraging path diversity, adapting to changing conditions, and optimizing resource utilization has a more direct role in overall performance in multipath transmission scenarios. Thus, in the following chapter, we concentrate on the design of the scheduling algorithm.

Chapter 4

Multipath Scheduling with Adversarial Multi-Armed Bandits

4.1 Introduction

Multipath transport-layer protocols like Multipath TCP (MPTCP) [29] and Multipath QUIC (MPQUIC)[10] allow the simultaneous use of various access technologies for quick and reliable data sharing, improving both transmission capacity and reliability over single-path alternatives. The sender distributes application data across available interfaces using a scheduling policy, and the receiver reassembles and reorders data from multiple paths transparently to the application. However, developing a multipath scheduler that can handle the varied features of network interfaces is challenging, especially when paths are heterogeneous in terms of latency and packet loss. This can result in packets arriving out of order, causing Head-of-Line (HoL) blocking or connection breakage. Additionally, the characteristics of paths are time-varying, making dynamic fluctuations in throughput common, especially in wireless networks. Such network heterogeneity and dynamic shifts are expected to become more prevalent in the next generation of networking technologies, which leads to the challenges of designing a scheduling policy adapting to the characteristics of future networks.

Multipath scheduling can be treated as a decision-making problem, which can be solved in a variety of ways. Round-Robin (RR) and Minimum Round-Trip-Time (MRTT) [29] are simple and easy to implement, however, their scheduling decisions do not consider the network dynamics. Calculation-based schedulers [12, 23] are pro-

posed with the consideration of some characteristics in MPTCP, but they assumed the transmission under a relatively stable network condition, which cannot effectively deal with network and wireless channel dynamics. Reinforcement Learning (RL) is also a possible strategy for scheduling problems. Alzadjali et al. [1] apply an online RL algorithm with a contextual multi-armed bandit (MAB) approach to MPTCP path management. Peekaboo [38] utilizes a contextual MAB in the scheduling strategy in MPQUIC, which incorporates a stochastic adjustment strategy to effectively learn the dynamic characteristics of the heterogeneous paths. However, based on the characteristics of the stochastic bandit model, these schedulers assume that the reward must follow some distributions, which creates a gap in the real situation. Thus, the shortcomings of current scheduling algorithms can be identified as follows: (i) performance degradation in the presence of network dynamics; (ii) strong assumptions and heavy training processes. How to deal with network heterogeneity and dynamics without relying on strong assumptions and heavy training processes is an open issue.

Motivated by the above observations, we intend to address the scheduling problem with a more accurate algorithm and fewer assumptions, taking into account both path heterogeneity and network dynamics, and minimizing the algorithm's training and response time. To achieve this goal, we propose a novel learning-based multipath scheduler with an adversarial multi-armed bandit (MSAB) to determine the scheduling strategy. Since the adversarial MAB model does not require the rewards to follow some distributions which is assumed in a stochastic MAB model [22], adversarial MAB would be a better match to the multipath scheduling problem. Considering the delay in receiving feedback on data transmission at the end host, this work employs an adversarial bandit model that accounts for delayed feedback. The contributions of this chapter can be summarised as follows:

- We formulate the multipath packet scheduling problem as an adversarial MAB problem with delayed feedback. Considering diverse characteristics across multiple paths, a multi-objective cost function is designed to optimize the performance of the scheduler under various kinds of network scenarios.

- We present a lightweight and deployable online learning strategy for generating a packet scheduling policy. Our proposed strategy enables the MSAB to operate without the need for an initial input or training process, facilitating efficient real-time multipath scheduling.

- We implement MSAB over our MPQUIC implementation in Chapter 3, and

conduct extensive experiments to evaluate its performance. It demonstrates the capacity to maintain lower completion time and higher goodput in heterogeneous and dynamic network scenarios, which significantly outperforms the state-of-the-art schedulers.

The remainder of this chapter is organized as follows. Sec. 4.2 summarizes the background and related work, introducing the multipath transport-layer protocols and the state-of-the-art schedulers. Sec. 4.3 presents the overall design of the MSAB scheduler. It elaborates on problem description and challenges, online learning strategy, cost function, and comprehensive algorithm in detail. To verify the performance gain of our proposal, experiments using ns-3 along with analyses are given in Sec. 4.4, followed by conclusions in Sec. 4.5.

4.2 Background and Related Work

In this section, we first go over the multipath transport-layer protocols. We then analyze the limitations of the state-of-the-art multipath schedulers that motivate our design of MSAB.

4.2.1 Multipath Transport-Layer Protocols

Today's end hosts are often equipped with more than one network interface, and it is desirable and advantageous to use and switch between them seamlessly. In order to achieve this, several multipath extensions to transport-layer protocols have been proposed. Concurrent Multipath Transfer for SCTP (CMT-SCTP) [18], MPTCP [29], and MPQUIC [10] are the most prevalent implementations. Among them, MPQUIC was chosen as the basis for assessing our algorithm since QUIC is the latest transport-layer protocol and therefore attracts much attention. MPQUIC overcomes a set of issues caused by MPTCP, e.g., connection breakage and HoL blocking. Benefiting from several specific features such as the 0-RTT handshake and stream multiplexing, MPQUIC is promising to satisfy the demands of future applications and next-generation networks. Nevertheless, our MSAB is also designed to be easily integrated into other multipath protocols.

4.2.2 Limitations of the State-of-the-Art Schedulers

Recent researches on the scheduling algorithms in multipath transport-layer protocols can be divided into: (1) heuristic-based approaches, e.g., RR and MRTT, (2) estimation-based models, e.g., [12, 23, 13, 46], and (3) learning-based policies, e.g., [44, 1, 38]. Detailed analysis is given as follows.

RR cyclically arranges packets on each path as long as it is allowed by the congestion window, but it does not employ any characteristics of the paths in the decision process. MRTT picks the path with the shortest RTT out of all available paths, which may always take the same path and lose the opportunity to explore other essential paths. These approaches do not need any additional analysis and calculations to make the scheduling decision. Obviously, they are easy to implement and require less processing time. However, their performance is often inferior under heterogeneous scenarios since the overall performance is constrained by the slow path.

BLEST [12] and ECF [23] estimate how many packets could be sent on the fast path during the RTT of the slow path. When the slow path becomes available, instead of immediately sending packets on the slow path, they might wait for the fast path to become available based on their estimations. DEMS [13] and DAMS [46] estimate RTT to arrange the sending order of packets with the consideration of the packets' deadline. However, the above solutions rely on many assumptions which may not always hold, so they are sensitive to sudden network condition changes in dynamic environments.

To deal with dynamic environments, learning-based approaches have been investigated. ReLeS [44] used a neural adaptive multipath scheduler based on deep reinforcement learning (DRL). It applied Deep Q-Network to teach the multipath scheduler from online data. However, DRL-based techniques increase the algorithm's reaction time dramatically [41]. Therefore, the lightweight MAB framework is more desirable. Alzadjali et al. [1] proposed an online MPTCP path manager based on the contextual MAB to help choose the primary path that maximizes throughput and minimizes delay and packet loss. A recent contextual MAB-based MPQUIC scheduler, Peekaboo [38], is designed to deal with dynamically changing channel conditions on heterogeneous paths. It applies LinUCB to process the online learning part, which assumes that the reward function is under some distributions.

Table 4.1: Summary of Key Notations

Notations	Description
t; T	The current time step; the total time.
i; K	The current path ID; the total number of paths.
$p_t^{(i)}$	The split ratio of sending packets for path i.
$w_t^{(i)}$	The congestion window (cwnd) of path i.
$u_t^{(i)}$	The number of inflight packets for path i.
r	The previous time step before t.
a_r	The path ID at previous time step r.
$d_r^{(a_r)}$	The RTT of path a_r.
$v_r^{(a_r)}$	The number of lost packets for path a_r.

4.3 Design of the MSAB Scheduler

Motivated by the aforementioned algorithms and their limitations in previous sections, we investigate the multipath scheduling problem as an adversarial bandit model. The detailed design of MSAB is presented in this section.

4.3.1 Problem Description and Challenges

A multipath scheduler determines which path to transmit each packet. In each time step t, the scheduler needs to determine the amount of data to be allocated to each path. An episode T is defined as the duration of an MPQUIC connection, starting from the initiation of the MPQUIC session until it is terminated, where $0 \leq t \leq T$. Assume an MPQUIC connection contains K paths, where $K \geq 1$. Let $p^{(i)}$ be the split ratio of packets delivered over the i-th path, and then we have $\sum_{i=1}^{K} p^{(i)} = 1$. The scheduler's goal is to find the best set of split ratios $P = (p^{(1)}, p^{(2)}, ..., p^{(K)})$ for the paths that lead to the optimal performance. During the process of data transmission in an MPQUIC connection, many important parameters can be collected to estimate the network conditions. Table 4.1 introduces the key notations that are used in our design.

In Sec. 4.2.2, we showed that the existing schedulers have their pros and cons. As a result, we need a multipath scheduler that can adapt to both heterogeneous and dynamic settings. The primary challenge we encounter is that the observed variables collected by the sender are considerably delayed. Until the successful receipt of the

acknowledgment, the sender is unable to estimate the transmission time of a path. We are limited to making predictions based on past observations. The second challenge pertains to a unique technique in MPQUIC, referred to as delayed ACK. To reduce overhead, it will not send an acknowledgment for every packet but will instead wait a while before sending an ACK packet that contains the acknowledgments for all previously received packets. It causes longer feedback delays and creates difficulties throughout the learning process. Additionally, most of the learning algorithm requires knowledge of the overall system but it is not possible for end hosts in network transmission. We have very limited information at our disposal. At the same time, we want our algorithm to be adaptive according to the network heterogeneity and dynamics.

4.3.2 Online Learning Strategy

We employ an online learning method based on MAB theory to handle the challenges mentioned in Sec. 4.3.1. For the online learning strategy, we need an approach that is: (i) accurate, to match the characteristics of paths, and (ii) lightweight in training and computation, to ensure the network performance is not adversely affected. In pursuit of this goal, we utilize a lightweight learning-based approach, employing an adversarial MAB model with delayed feedback to generate a packet scheduling policy. Our approach stands apart from heuristic techniques that rely on fixed control rules derived from simplified models of the deployment environment or complex reinforcement learning with extended training periods and stringent assumptions. Rather, our techniques attempt to learn a packet scheduling policy from observations, which is more adaptive to the varying of network conditions and traffic patterns. Our model is defined as follows.

Agent. A learning agent is a system entity that performs a learning task, which observes a vector of features that reflect the surrounding environment. The goal of the agent is to minimize a cost function (or maximize a reward function) that measures how well the agent adapts to the environment by choosing an action from a set of options. In the multipath packets scheduling problem, the agent is responsible for determining the traffic allocation over multiple paths on the sender side of a connection. Specifically, we adopt Exp3 [5] to take the observations from the environment as input, and derive a distribution of the split ratios P that represents the ratios of data to be allocated on the set of paths.

Arm. Arms are the alternates that an agent chooses. In our problem, each path is treated as an arm i. We have a total of K arms.

State. A state of the system represents the information of a snapshot of the environment that an agent observes. The agent monitors the system state at the start of each time step. Assume the system state is represented by $\boldsymbol{s}_t = (s_t^{(1)}, s_t^{(2)}, \ldots, s_t^{(K)})$ at the t-th time step, where $s_t^{(i)}$ represents the observed state of the i-th path in multipath transmission. To reflect the network conditions more accurately, the state of the multipath scheduler consists of both the current state at the t-th time step and the delayed feedback of some previous time steps r but received at time step t. The current state $s_t^{(i)}$ can be represented by a tuple $(w_t^{(i)}, u_t^{(i)})$, where $w_t^{(i)}$ is the cwnd and $u_t^{(i)}$ is the number of inflight packets for path i. The delayed feedback $q_r^{a_r}$ is popped from a queue Q_t, where $q_r^{a_r} \in Q_t$ and $q_r^{a_r} = (r, a_r, d_r^{(a_r)}, v_r^{(a_r)})$. Here, $d_r^{(a_r)}$ represents the latest RTT and $v_r^{(a_r)}$ represents the number of lost packets, where a_r is the path ID at the r-th time step.

Cost. At each time step, the agent observes a state $\boldsymbol{s}_t$ and takes an action. After applying the action, the state of the environment transitions to $\boldsymbol{s}_{t+1}$ and some cost incurred for taking the action. The cost at the t-th time step from path i is $l_t^{(i)} \in [0, 1]$, and let system cost $\boldsymbol{l}_t = (l_t^{(1)}, l_t^{(2)}, \ldots, l_t^{(K)})$. The detailed cost function used in our system is discussed in Sec. 4.3.3.

Action. An action describes how an agent reacts to a given situation. In our problem, an action is a scheduling choice, which determines how to distribute the current traffic over multiple paths. An action can be represented by a vector $\boldsymbol{p}_t = (p_t^{(1)}, p_t^{(2)}, \ldots, p_t^{(K)})$, indicating that the packets will be delivered over the i-th path according to the distribution $\boldsymbol{p}_t$. The action depends on the costs from both the current state and delayed feedback. After obtaining the costs $l_t^{(i)}$ for path i, we are able to update the weights $\tilde{L}_t^{(i)}$ of path i, where

$$\tilde{L}_t^{(i)} = \tilde{L}_{t-1}^{(i)} + \eta_e \frac{l_t^{(i)}}{p_t^{(i)}}, \tag{4.1}$$

where η_e is the step size. By Exp3 [5], we can get the distribution of actions $\boldsymbol{p}_t$ for the next step, where

$$\boldsymbol{p}_{t+1}^{(i)} := \frac{e^{-\tilde{L}_t^{(i)}}}{\sum_{j=1}^{n} e^{-\tilde{L}_t^{(j)}}}. \tag{4.2}$$

Then, we distribute the packets to each path proportionally according to the distri-

bution $\boldsymbol{p}_t$.

In order to ensure adaptability to network conditions and mitigate the impact of delayed feedback on the algorithm, we employ an adaptive step size adjustment mechanism that adheres to the following constraints. The influence of the delayed feedback is based on the number of missing feedback samples m_t so far, where

$$m_t = t - \sum_{\tau=1}^{t} |Q_\tau| \,. \tag{4.3}$$

If the number of missing samples is too large (i.e. $\sum_{\tau=1}^{t} m_\tau \geq 2^e$), we will increase the epoch index by $e = e + 1$ and initialize $\tilde{L}_t^{(i)} = 0$ for $i = 1, ..., K$. Then, we update the weights of path i using step size

$$\eta_e = \sqrt{\frac{\ln K}{2^e}}. \tag{4.4}$$

The epoch index and the step size are proposed by [5], which uses a doubling trick to deal with the unknown T and delayed feedback. Feedback samples originating in the previous epoch are discarded once received.

Regret. A regret indicates the discrepancy between the minimal and actual costs. In the multipath packet scheduling problem, our regret is

$$R(T) = \sum_{t=1}^{T} l_t - \min \sum_{t=1}^{T} l_t. \tag{4.5}$$

Given the above preliminaries, the multipath packet scheduling problem can be represented by an adversarial bandit model with delayed feedback: learning an optimal policy to minimize the expected cumulative cost during the data transmission.

4.3.3 Cost Function

At the end of each time step, the agent evaluates the performance of the system using a cost function that transforms the statistics gathered at that time step into a numerical utility value. The cost is a complex function because the scheduler needs to consider a variety of characteristics to effectively leverage multipath scenarios.

To reflect the network conditions accurately and comprehensively, the cost function here considers both the current system state at time step t and the delayed

feedback which reflects the path conditions at time step r. Our scheduler adopts the following cost function to assess the performance of the scheduling action:

$$l_t^{(i)} = C_t^{\mathrm{RTT}(i)} + \alpha C_t^{\mathrm{Lost}(i)} + \beta C_t^{\mathrm{Used}(i)}, \tag{4.6}$$

where $0 < \alpha < 1$, $0 < \beta < 1$ are weights.

$C_t^{\mathrm{RTT}(i)} = \sum_{r \in Q_t} d_r^{(a_r)} / |Q_t|$ is the average of RTT of the received feedback in Q_t. $d_r^{(a_r)}$ represents the RTT of path a_r that is measured by the acknowledgement packets. Minimizing this term will minimize the average delay of all packets and reduce the completion time. By minimizing this term, the scheduler is in favour of low-latency paths.

$C_t^{\mathrm{Lost}(i)} = \sum_{r \in Q_t} v_r^{(a_r)}$ is the total number of retransmitted packets. Normally it is proportional to the number of lost packets and reflects the network congestion conditions. By minimizing this term, the scheduler is in favour of less error-prone or congested paths, which can reduce packet loss rate and ease the network congestion conditions.

$C_t^{\mathrm{Used}(i)} = u_t^{(i)} / w_t^{(i)}$ is the ratio of the used sending buffer, where $u_t^{(i)}$ indicates the number of inflight packets (unacknowledged packets) for path i. Minimizing $C_t^{\mathrm{Used}(i)}$ can reduce bufferbloat and increase the throughput. By minimizing this term, the scheduler is in favour of high-throughput paths.

The cost function $l_t^{(i)}$ defines the cost of path i at time step t. In the adversarial bandit model, the objective is to minimize the expected regret, indicating minimizing the cumulative cost.

4.3.4 Overall Algorithm

We now present the overall structure of the MSAB scheduler shown in Figure 4.1. The sender gets the data from the application and allocates them to different paths based on the scheduling strategy advised by the adversarial bandit algorithm. Paths are connected to the core networks and transmit packets to the receiver side. The receiver then sends out the acknowledgment of the received packets, which contain the information for the delayed feedback. With the information of RTT and retransmitted packets from the acknowledgment and the used buffer size from the sender, our algorithm is able to get the transmission cost and update its scheduling strategy for the following decisions.

Algorithm 4 presents the MSAB step-by-step. The number of paths is the input

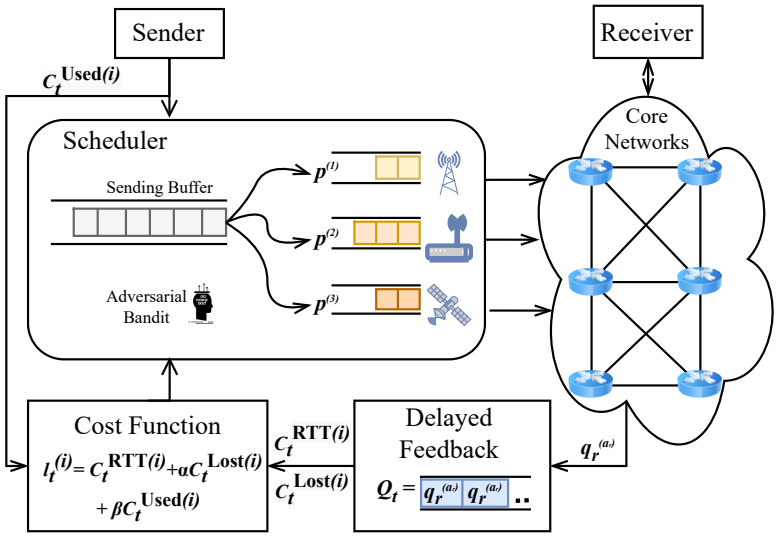


Figure 4.1: Overall Structure of MSAB.

of the algorithm. The scheduler will first distribute packets equally for each path to get the initial cost. During the transmission process, MSAB consists of two parts. First is the delayed feedback-receiving process (Line 16) which will be called when an acknowledgment is received. It grabs the information from the ACK frame and pushes the required information into the feedback queue, including the path, the latest RTT, and the number of lost packets. The major part of MSAB is the scheduling part (Lines 1-15) which observes the new state of each path for each time step and pops a set of delayed feedback from the queue. After that, it will adaptively adjust the step size based on the size of Q_t and calculate the cost for each path. Then, the weight of paths could be updated as well as the distribution of the scheduling strategy. The new distribution then is able to be applied in the next step of packet arrangement.

(a) Two paths (b) Four paths

Figure 4.2: Experimental topologies.

Algorithm 4 MSAB Scheduler

Input: K
Initial: $\tilde{L}_1^{(i)} = 0$, $e = 0$, $\eta_0 = 1$, and $p_0^{(i)} = \frac{1}{K}$.
1: Arrange packets on paths according to p_0.
2: **for** $t = 1, ..., T$ **do**
3: Get the current state s_t.
4: Pop a set of delayed feedback from queue Q_t.
5: $m_t = t - \sum_{\tau=1}^{t} |Q_\tau|$.
6: **if** $\sum_{\tau=1}^{t} m_\tau \geq 2^e$ **then**
7: $e = e + 1$.
8: $\tilde{L}_1^{(i)} = 0$ for $i = 1, ..., K$.
9: **end if**
10: $\eta_e = \sqrt{\frac{\ln K}{2^e}}$.
11: $l_t^{(i)} = C_t^{\mathrm{RTT}(i)} + \alpha C_t^{\mathrm{Lost}(i)} + \beta C_t^{\mathrm{Used}(i)}$.
12: $\tilde{L}_t^{(i)} = \tilde{L}_{t-1}^{(i)} + \eta_e \frac{l_t^{(i)}}{p_t^{(i)}}$.
13: $p_t^{(i)} = \frac{e^{-\tilde{L}_t^{(i)}}}{\sum_{j=1}^{n} e^{-\tilde{L}_t^{(j)}}}$.
14: Arrange packets on paths according to p_t.
15: **end for**

Delayed feedback received
Input: $r, a_r, d_r^{(a_r)}, v_r^{(a_r)}$
16: Push the delayed feedback into queue Q_t.

4.4 Evaluation

We implement the proposed MSAB[1] on top of the MPQUIC implementation in ns-3 [34], in which we can assess the performance of the proposed algorithm with a wide range of network configurations. We investigate its performance by comparing it with four state-of-the-art multipath scheduling algorithms, RR, MRTT, ECF [23], and Peekaboo [38].

4.4.1 Testbed Construction and Experiment Setup

To reproduce the network environment of heterogeneity and the scalability of MSAB, we construct two topologies in Figure 4.2, which has multiple paths between the client and the server. Each path is characterized independently by its bandwidth, OWD,

[1]https://github.com/ssjShirley/mpquic-ns3/tree/mpquic-msab

and loss rate. We generate multiple configurations to evaluate the performance with respect to other schedulers and consider both dominating and competing scenarios to evaluate the adaptivity of our proposed algorithm. Table 4.2 shows the basic path setting of a dominating or competing scenario with the two-path topology shown in Figure 4.2a. In the dominating scenario, P0 has lower bandwidth and higher OWD, while P1 has higher bandwidth and lower OWD, bringing a heterogeneous but dominating scenario as P0 would normally be faster than P1. In the competing scenario, P0 has lower bandwidth and lower OWD, while P1 has higher bandwidth and higher OWD, so it is hard to determine which path is faster than the other, bringing a heterogeneous and competing scenario.

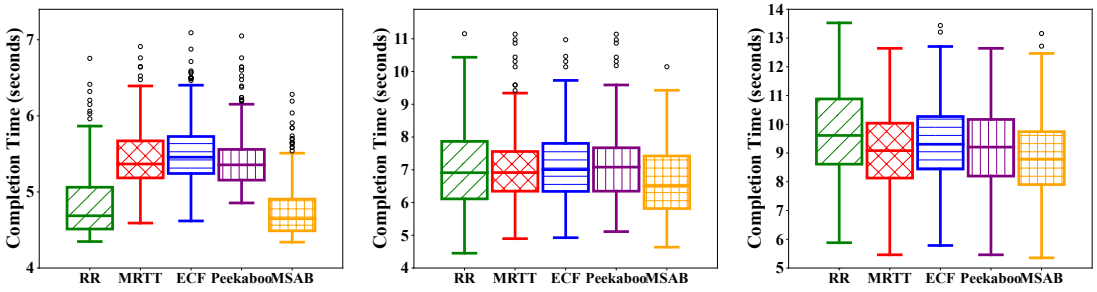
Table 4.2: The Range of Path Parameters

Scenarios	Path	Bandwidth	OWD	Loss
Dominating	P0	5–6Mbps	50–55ms	0–0.5%
	P1	10–11Mbps	10–11ms	0–0.5%
Competing	P0	5–6Mbps	10–11ms	0–0.5%
	P1	10–11Mbps	50–55ms	0–0.5%

Table 4.3: The Variance of Simulation Parameters

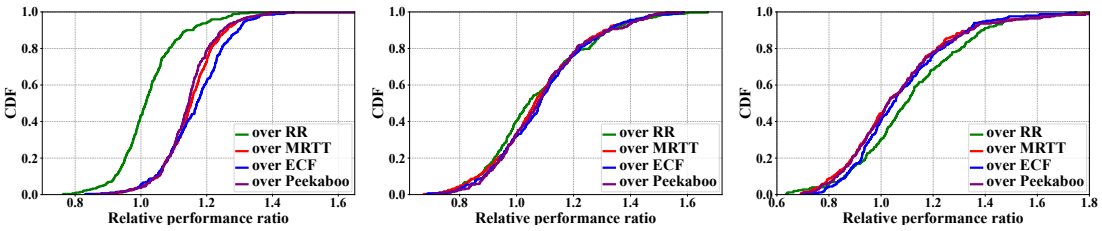
Dynamic Level	Low	Medium	High
Bandwidth Variance [%]	1	3	5
OWD Variance [%]	1	5	10
Loss variance [%]	0.1	0.3	0.5

To evaluate MSAB in a wider range of configurations and show its effectiveness under different settings, we distribute the design parameters equally across the experiments. The bandwidth and OWD of paths are changed every 0.1 seconds under a uniform random sampling [27]. The range of the uniform random sampling depends on the variance in Table 4.3, which differs from the dynamic levels. To ensure statistically significant results, for each path configuration, we run 100 repetitions for each scheduler. We consider file download applications and the file size is 5 MB in our evaluation. For each dynamic level, we evaluate both the download completion time and the instantaneous goodput (i.e., the application-level throughput). some typical results are analyzed in the next few sections.



(a) Low dynamic (b) Medium dynamic (c) High dynamic

Figure 4.3: Completion time of different dynamic levels in the dominating scenario.

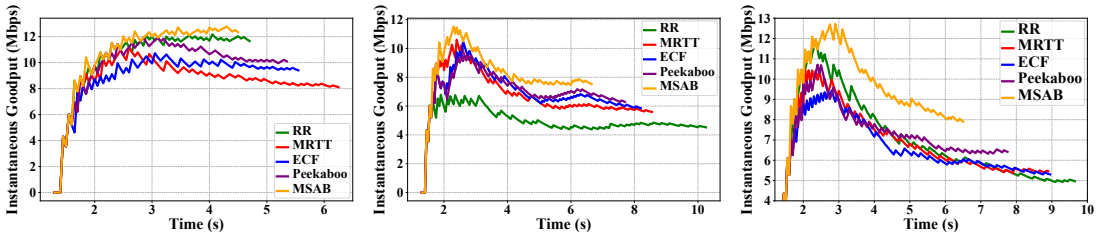


(a) Low dynamic (b) Medium dynamic (c) High dynamic

Figure 4.4: CDF of different dynamic levels in the dominating scenario.

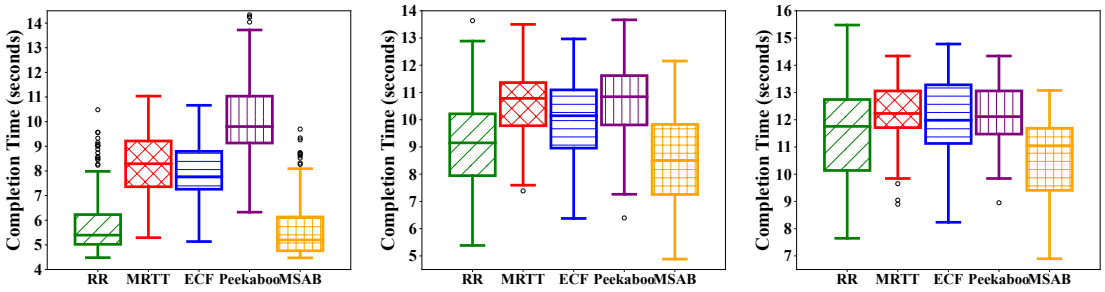
4.4.2 Performance of Dominating Scenarios

We first illustrate the median file transmission completion times in dominating scenarios with the two-path topology (Figure 4.2a). Figures 4.3a, 4.3b, and 4.3c present that MSAB has a lower completion time in all dynamic levels when downloading a fixed-size file. RR has a better performance in the low dynamic scenario while MRTT and Peekaboo perform better, which reflects that other schedulers cannot maintain a good performance in different dynamic scenarios. Figure 4.4 presents the Cumulative Distribution Function (CDF) of the performance improvement of MSAB at all dynamic levels, which reflects that MSAB is able to outperform all other schedulers at all dynamic levels in over 60% of the path configurations in the dominating scenarios. Further, the gains are over 20% for more than 20% of the path configurations in the high dynamic level. Figure 4.5 depicts the instantaneous goodput of a typical transmission, and we observe that MSAB has higher instantaneous goodput over time compared to other schedulers.



(a) Low dynamic (b) Medium dynamic (c) High dynamic

Figure 4.5: Instantaneous goodput of different dynamic levels in the dominating scenario.

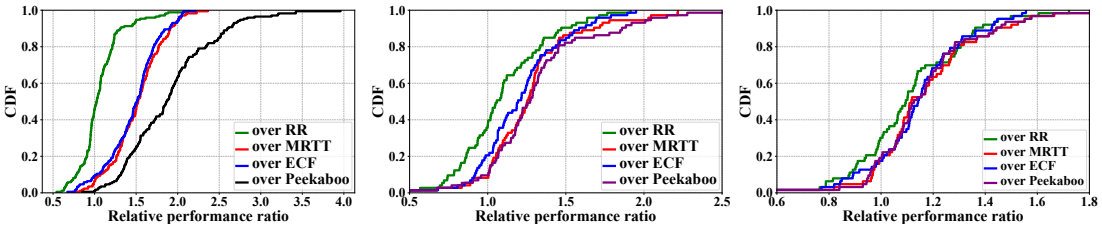


(a) Low dynamic (b) Medium dynamic (c) High dynamic

Figure 4.6: Completion time of different dynamic levels in the competing scenario.

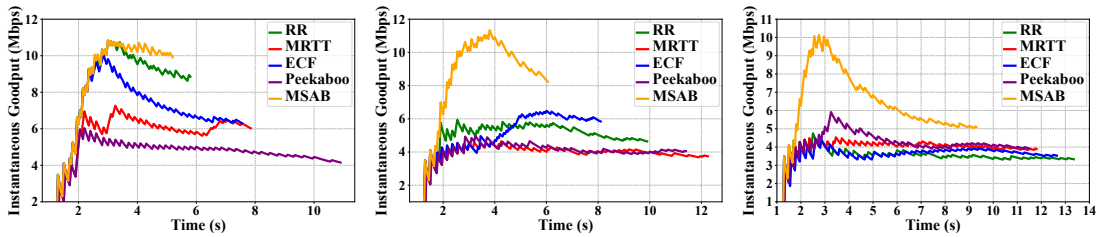
4.4.3 Performance of Competing Scenarios

In competing scenarios, we similarly evaluate the performance in different dynamic levels and analyze the transmission at the high dynamic level. For each dynamic level, Figures 4.6a, 4.6b, and 4.6c illustrate the median file download completion times for different schedulers. MSAB has a lower range of completion time in all dynamic levels and significantly outperforms the other schedulers in the high dynamic scenario. Figure 4.7 presents the CDF of the performance improvement of MSAB at all dynamic levels. MSAB significantly outperforms MRTT, ECF, and Peekaboo in 90% of the path configurations at the low dynamic level. We also observe that MSAB outperforms other schedulers in over 70% of the path configurations with 20% overhead for more than 30% of the samples in competing scenarios at the high dynamic level from the CDF in Figure 4.7c. Figure 4.8 shows that MSAB has a significantly high instantaneous goodput for a typical transmission over time.



(a) Low dynamic (b) Medium dynamic (c) High dynamic

Figure 4.7: CDF of different dynamic levels in the competing scenario.

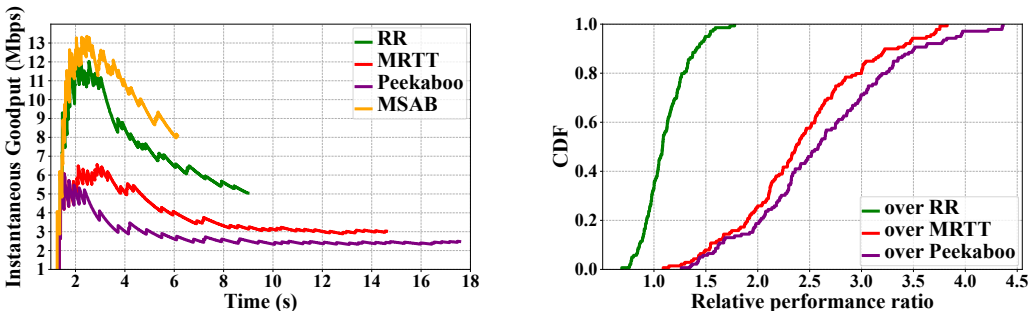


(a) Low dynamic (b) Medium dynamic (c) High dynamic

Figure 4.8: Instantaneous goodput of different dynamic levels in the competing scenario.

4.4.4 Performance of Four Paths Scenarios

To present the scalability of MSAB, we also evaluate the performance with a four-path topology (Figure 4.2b). Each path has a range of 5-6 Mbps bandwidth, 10-12 ms OWD, and 0.05% loss rate under a uniform random sampling. As ECF is only designed for two paths, we here only compare MSAB with RR, MRTT, and Peekaboo. Figure 4.9a illustrates the instantaneous goodput of a typical transmission of 5 MB data and presents that MSAB has a significantly higher overhead in this scenario, which illustrates that MSAB is able to simultaneously utilize the bandwidth of multiple paths and is able to reach a lower transmission complete time. Figure 4.9b presents the CDF of the performance improvement of MSAB over other schedulers and we observe that MSAB outperforms MRTT and Peekaboo in over 90% of the path configurations and with 100% overhead for more than 80% of the samples with a four-path topology. As MSAB is able to transmit on multiple paths simultaneously and gain the aggregated bandwidth, it gets a large performance gain when it comes with more than two paths.



(a) Instantaneous goodput (b) CDF

Figure 4.9: Instantaneous goodput and CDF in the four paths scenario.

4.5 Summary

The multipath scheduler is a fundamental mechanism that has a significant impact on the performance of MPQUIC. To cope with the challenges of network heterogeneity, comprehensive QoS goals, and dynamic environments, we proposed a novel adaptive multipath scheduler called MSAB. MSAB consists of three key designs: (1) An adversarial bandit model to formulate the multipath packet scheduling problem with delayed feedback; (2) A comprehensive cost function to consider diverse characteristics for performance optimization; and (3) A lightweight and deployable online learning strategy to generate a packet scheduling policy. We implemented MSAB in ns-3 and evaluated it over different dynamic levels of network conditions, which showed that MSAB significantly outperformed the state-of-the-art schedulers in heterogeneous scenarios. Future research will take into account verifying the MSAB's performance with a larger variance of dynamic variable settings, and in large-scale network or real-world scenarios.

Chapter 5

Conclusions

In this thesis, we first presented an ns-3 implementation of MPQUIC in Chapter 3, which overcame the challenges of advertising multiple addresses, separating transmission paths, and extending the scheduling and congestion control algorithms, while still maintaining the original transmission features. With a set of experimentations, we evaluated the correctness of our implementation, the scalability of multiple paths, the flexibility of the path schedulers, and the congestion control algorithms, which provide a stable simulation platform for the research community on multipath transport protocols.

On the other hand, the multipath scheduler is a fundamental mechanism that has a significant impact on the performance of MPQUIC. To cope with the challenges of network heterogeneity, comprehensive QoS goals, and dynamic environments, we further proposed a novel adaptive multipath scheduler in Chapter 4. MSAB consists of three key designs: (1) An adversarial bandit model to formulate the multipath packet scheduling problem with delayed feedback; (2) A comprehensive cost function to consider diverse characteristics for performance optimization; and (3) A lightweight and deployable online learning strategy to generate a packet scheduling policy. We implemented MSAB in ns-3 and evaluated it over the different dynamic levels of network conditions, which showed that MSAB significantly outperformed the state-of-the-art schedulers in heterogeneous scenarios.

A number of places can be improved in the future. We will further implement the currently missing features in MPQUIC ns-3 according to the IETF draft. Future research will take into account verifying the MSAB's performance with frequently fluctuating bandwidth and in large-scale network scenarios. Exploring better scheduling and congestion control algorithms will also be considered in future work.

Bibliography

[1] Aziza Alzadjali, Flavio Esposito, and Jitender Deogun. A contextual bi-armed bandit approach for MPTCP path management in heterogeneous LTE and WiFi edge networks. In *Proceedings of 2020 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 307–316, 2020.

[2] Apple. Safari 14 release notes, 2020.

[3] Donald A Berry and Bert Fristedt. Bandit problems: sequential allocation of experiments (monographs on statistics and applied probability). *London: Chapman and Hall*, 5(7):71–87, 1985.

[4] Mike Bishop. HTTP/3. RFC 9114, 2022.

[5] Ilai Bistritz, Zhengyuan Zhou, Xi Chen, Nicholas Bambos, and Jose Blanchet. Online EXP3 learning in adversarial bandits with delayed feedback. In *Proceedings of Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[6] Giuseppe Burtini, Jason L. Loeppky, and Ramon Lawrence. A survey of online experiment design with the stochastic multi-armed bandit. *CoRR*, abs/1510.00757, 2015.

[7] Stuart Cheshire, David Schinazi, and Christoph Paasch. Advances in networking, part 1. 2017.

[8] Dragana Damjanovic. QUIC and HTTP/3 support now in firefox nightly and beta, 2021.

[9] Alvise De Biasio, Federico Chiariotti, Michele Polese, Andrea Zanella, and Michele Zorzi. A QUIC implementation for ns-3. In *Proceedings of the 2019 Workshop on Ns-3 (WNS3)*, page 1–8, 2019.

[10] Quentin De Coninck and Olivier Bonaventure. Multipath QUIC: Design and evaluation. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, page 160–166, 2017.

[11] Quentin De Coninck and Olivier Bonaventure. Multiflow QUIC: A generic multipath transport protocol. *IEEE Communications Magazine*, 59(5):108–113, 2021.

[12] Simone Ferlin, Özgü Alay, Olivier Mehani, and Roksana Boreli. BLEST: Blocking estimation-based MPTCP scheduler for heterogeneous networks. In *Proceedings of 2016 IFIP Networking Conference (IFIP Networking) and Workshops*, pages 431–439, 2016.

[13] Yihua Ethan Guo, Ashkan Nikravesh, Z. Morley Mao, Feng Qian, and Subhabrata Sen. Accelerating multipath transport through balanced subflow completion. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking (MobiCom)*, page 141–153, 2017.

[14] Sangtae Ha, Injong Rhee, and Lisong Xu. CUBIC: A new TCP-friendly high-speed TCP variant. *SIGOPS Operating Systems Review*, 42(5):64–74, 2008.

[15] Ryan Hamilton. First chromium code landing: Cl 11125002: Add quicframer and friends, 2012.

[16] David Hayes, David Ros, Lachlan L.H. Andrew, and Sally Floyd. Common TCP evaluation suite. draft-irtf-iccrg-tcpeval-01, IETF, 2014.

[17] Jana Iyengar and Martin Thomson. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000, 2021.

[18] J.R. Iyengar, P.D. Amer, and R. Stewart. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths. *IEEE/ACM Transactions on Networking*, 14(5):951–964, 2006.

[19] P. Karn and C. Partridge. Improving round-trip time estimates in reliable transport protocols. *SIGCOMM Computer Communication Review*, 17(5):2–7, 1987.

[20] Ramin Khalili, Nicolas Gast, Miroslav Popovic, and Jean-Yves Le Boudec. MPTCP is not pareto-optimal: Performance issues and a possible solution. *IEEE/ACM Transactions on Networking*, 21(5):1651–1665, 2013.

[21] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The QUIC transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM)*, page 183–196, 2017.

[22] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

[23] Yeon Lim, Erich M. Nahum, Don Towsley, and Richard J. Gibbens. ECF: An MPTCP path scheduler to manage heterogeneous paths. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT)*, page 147–159, 2017.

[24] Yanmei Liu, Yunfei Ma, Quentin De Coninck, Olivier Bonaventure, Christian Huitema, and Mirja Kühlewind. Multipath Extension for QUIC. Internet-Draft draft-ietf-quic-multipath-04, Internet Engineering Task Force, 2023.

[25] Jayaram Mudigonda, Praveen Yalagandula, Mohammad Al-Fares, and Jeffrey C. Mogul. SPAIN: COTS Data-Center ethernet for multipathing over arbitrary topologies. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, page 18, 2010.

[26] Christoph Paasch, Gregory Detal, Fabien Duchene, Costin Raiciu, and Olivier Bonaventure. Exploring mobile/WiFi handover with multipath TCP. In *Proceedings of the 2012 ACM SIGCOMM Workshop on Cellular Networks: Operations, Challenges, and Future Design (CellNet)*, page 31–36, 2012.

[27] Christoph Paasch, Ramin Khalili, and Olivier Bonaventure. On the benefits of applying experimental design to improve multipath TCP. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, page 393–398, 2013.

[28] Umberto Paro, Federico Chiariotti, Anay Ajit Deshpande, Michele Polese, Andrea Zanella, and Michele Zorzi. Extending the ns-3 QUIC module. In *Proceedings of the 23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, page 19–26, 2020.

[29] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath TCP. In *Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM)*, page 266–277, 2011.

[30] Dipankar Raychaudhuri, Kiran Nagaraja, and Arun Venkataramani. Mobility-first: A robust and trustworthy mobility-centric architecture for the future internet. *SIGMOBILE Mobile Computing and Communications Review*, 16(3):2–13, 2012.

[31] Eric Rescorla and Nagendra Modadugu. Datagram Transport Layer Security Version 1.2. RFC 6347, 2012.

[32] Michael Scharf and Sebastian Kiesel. Head-of-line blocking in TCP and SCTP: Analysis and measurements. In *Proceedings of the 2006 IEEE Global Communications Conference (GLOBECOM)*, pages 1–5, 2006.

[33] Xiang Shi, Lin Wang, Fa Zhang, Biyu Zhou, and Zhiyong Liu. Pstream: Priority-based stream scheduling for heterogeneous paths in multipath-QUIC. In *Proceedings of the 29th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–8, 2020.

[34] Shengjie Shu, Wenjun Yang, Jianping Pan, and Lin Cai. A multipath extension to the QUIC module for ns-3. In *Proceedings of 2023 Workshop on ns-3 (WNS3)*, 2023.

[35] Tobias Viernickel, Alexander Froemmgen, Amr Rizk, Boris Koldehofe, and Ralf Steinmetz. Multipath QUIC: A deployable multipath transport protocol. In *Proceedings of the IEEE International Conference on Communications (ICC)*, pages 1–7, 2018.

[36] Richard Weber. On the gittins index for multiarmed bandits. *The Annals of Applied Probability*, pages 1024–1033, 1992.

[37] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 99–112, 2011.

[38] Hongjia Wu, Özgü Alay, Anna Brunstrom, Simone Ferlin, and Giuseppe Caso. Peekaboo: Learning-based multipath scheduling for dynamic heterogeneous environments. *IEEE Journal on Selected Areas in Communications*, 38(10):2295–2310, 2020.

[39] Hongjia Wu, Giuseppe Caso, Simone Ferlin, Özgü Alay, and Anna Brunstrom. Multipath scheduling for 5G networks: Evaluation and outlook. *IEEE Communications Magazine*, 59(4):44–50, 2021.

[40] Jianpeng Xu and Bo Ai. Deep reinforcement learning for handover-aware MPTCP congestion control in space-ground integrated network of railways. *IEEE Wireless Communications*, 28(6):200–207, 2021.

[41] Francis Y. Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In *Proceedings of 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 495–511, 2020.

[42] Wenjun Yang, Lin Cai, Shengjie Shu, and Jianping Pan. Scheduler design for mobility-aware multipath QUIC. In *Proceedings of 2022 IEEE Global Communications Conference (GLOBECOM)*, pages 2849–2854, 2022.

[43] Wenjun Yang, Shengjie Shu, Lin Cai, and Jianping Pan. MM-QUIC: A mobility-aware multipath QUIC for satellite networks. In *Proceedings of the 17th International Conference on Mobility, Sensing and Networking (MSN)*, pages 608–615, 2021.

[44] Han Zhang, Wenzhong Li, Shaohua Gao, Xiaoliang Wang, and Baoliu Ye. ReLeS: A neural adaptive multipath scheduler based on deep reinforcement learning. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pages 1648–1656, 2019.

[45] Lixia Zhang. Why TCP timers don't work well. *SIGCOMM Computer Communication Review*, 16(3):397–405, 1986.

[46] Xutong Zuo, Yong Cui, Xin Wang, and Jiayu Yang. Deadline-aware multipath transmission for streaming blocks. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pages 2178–2187, 2022.