A Multipath Extension to the QUIC Module for ns-3

Shengjie Shu
University of Victoria
Victoria, BC, Canada
shengjies@uvic.ca

Wenjun Yang
University of Victoria
Victoria, BC, Canada
wenjunyang@uvic.ca

Jianping Pan
University of Victoria
Victoria, BC, Canada
pan@uvic.ca

Lin Cai
University of Victoria
Victoria, BC, Canada
cai@ece.uvic.ca

ABSTRACT

Network transmission with multiple interfaces is desirable for the next-generation Internet to improve end-to-end performance and reliability. Multipath QUIC is thus proposed to utilize multiple interfaces for Internet transmission with QUIC, while QUIC is already standardized and active in use by several 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. In this paper, we present our implementation and improvement of MPQUIC based on the QUIC module for ns-3, along with a description of the features that we have implemented. We also demonstrate the performance of MPQUIC using an expanding series of experiments under various scenarios. Our implementation meets the demands for scalable multiple paths, flexible path schedulers, and compatible congestion control algorithms.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Network simulations**; *Network protocol design.*

KEYWORDS

Multipath QUIC, network simulation, ns-3, transport protocols

ACM Reference Format:
Shengjie Shu, Wenjun Yang, Jianping Pan, and Lin Cai. 2023. A Multipath Extension to the QUIC Module for ns-3. In *2023 Workshop on ns-3 (WNS3 2023), June 28–29, 2023, Arlington, VA, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3592149.3592803

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 [14] 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 [1]. Nevertheless, the next-generation networks pose a set of challenges to the protocols built upon the TCP/IP stack, e.g., connection breakage [15], and Head-of-Line (HoL) blocking issue [16]. To address the problems with TCP, Google first proposed Quick UDP Internet Connections (QUIC) [9], which was later standardized by the Internet Engineering Task Force (IETF) [7] 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 [17]) 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, MPQUIC [3] 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 [4], protocol implementations [18], scheduling strategies [20], and congestion control algorithms [22]. 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 [2, 13] 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

WNS3 2023, June 28–29, 2023, Arlington, VA, USA

ACM ISBN 979-8-4007-0747-6/23/06…$15.00
https://doi.org/10.1145/3592149.3592803

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

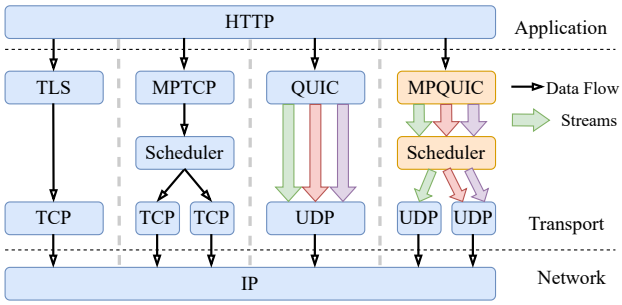


Figure 1: Structure of MPQUIC in Comparison with Others

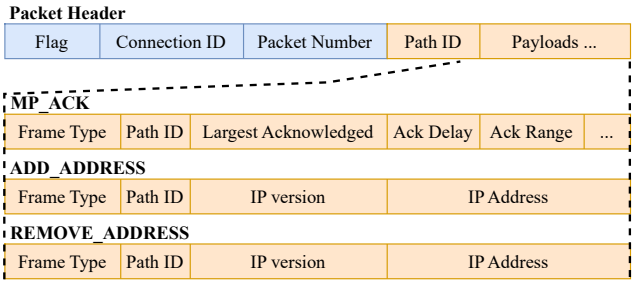


Figure 2: MPQUIC Header and New Frames

several bugs in the initial release and further enriched the set of path schedulers. Our implementation has been utilized in some research works [21, 22] since the first release.

In the remainder of this paper, we first outline the key components of the MPQUIC protocol as well as the difficulties encountered during the code development in Section 2. Then, we detail the implementation of MPQUIC over the QUIC module for ns-3 in Section 3. In Section 4 we expand the evaluation of the performance of our MPQUIC implementation. Finally, Section 5 concludes the paper and discusses future work.

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 [3]. The layered structure of MPQUIC is illustrated in Figure 1. Differing from QUIC that 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 [14] 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.

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 [12]. The single-path QUIC integrates the features of TCP, TLS, and a portion of HTTP/2 over the UDP transmission [9]. 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 [3, 18] 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 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 MPQUIC. Figure 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 [14].

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 [5] and ECF [10] schedule packets by estimating how many packets could be sent on the fast path during the RTT of the slow path. Peekaboo [20] 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 and CUBIC [6] are widely used for single-path TCP and QUIC, but they will cause unfairness in multipath protocols [19]. Thus, the default congestion control algorithm in MPQUIC incorporates OLIA [8],

which integrates the information from all paths and also presents a good performance in MPTCP.

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 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. As the diagram shows, the primary classes for MPQUIC are developed on top of the QUIC module [2] in ns-3. New classes, functions, and variables have been introduced to MPQUIC to account for the multipath features.

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) 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 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.

3.2 MPQUIC Headers and Frames

In order to distinguish packets transmitted over different paths, a new variable `m_pathId` is defined in the `QuicHeader`. Figure 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 [7], the existing QUIC module does not construct the functions to process these frames. As they are

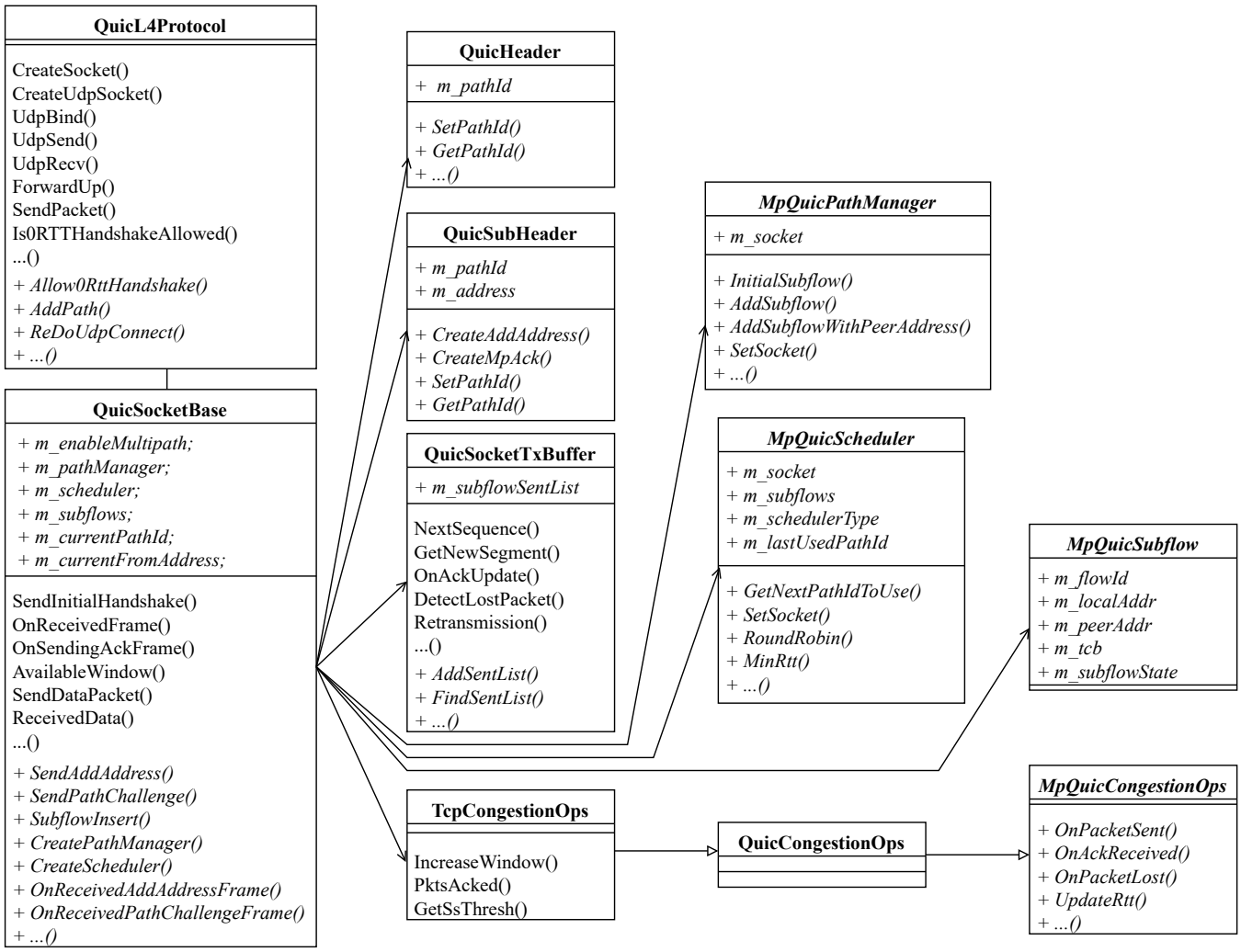


Figure 3: MPQUIC UML Diagram (New Classes, Functions, and Variables Shown in Italics)

required to address migration and multipath implementation, we also define `OnReceivedPathChallengeFrame()` and `OnReceivedPathResponseFrame()` in `QuicSocketBase` to handle these two frames.

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 this paper. 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 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

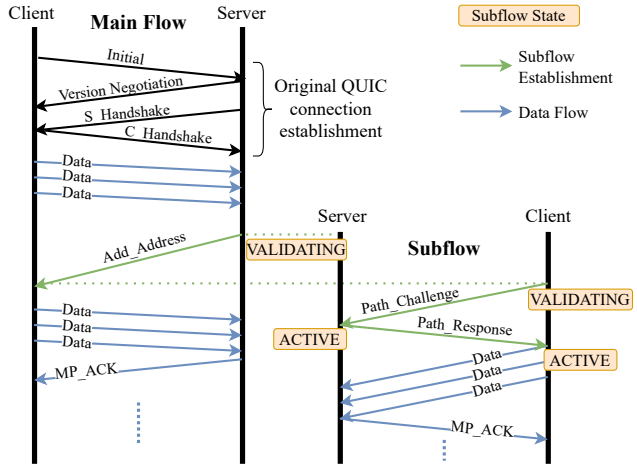


Figure 4: Procedures for Subflow Establishment

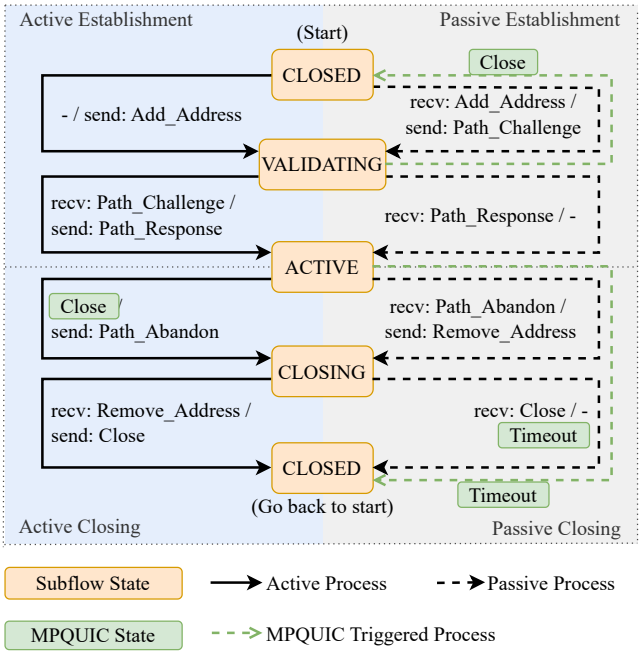


Figure 5: State Machine of a Subflow

authenticate the new client subflow without a three-way handshake.

The state machine of subflow maintenance is shown in Figure 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.

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

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 [2]. 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_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.

3.6 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`. `OnReceivedAckFrame()` 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.

3.7 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.

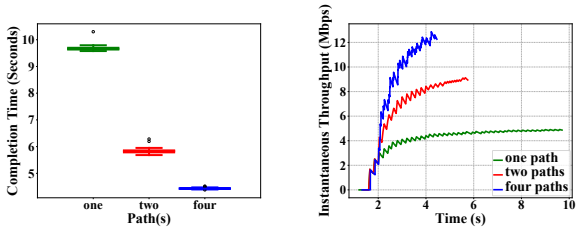
3.8 Current Status and Missing Features

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 [11]. Our implementation supports most of the features mentioned in [11], including handshake negotiation and transport parameters (Section 3.2), path setup and removal (Section 3.3), multipath operation with multiple packet number spaces (Section 3.6), path scheduling (Section 3.4), and congestion control (Section 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. We also enriched the set of path schedulers since then.

Here are some missing features mentioned in [11] that we will consider for our future work. [11] 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 5 shows, however, we do not support transmitting the path status between peers currently. In the path close stage, [11] 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, [11] 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.

(a) Four Paths (b) Two Paths

Figure 6: Topology with Multiple Paths



(a) Completion Time (b) Instantaneous Throughput

Figure 7: Performance Comparison with One, Two, and Four Paths

4 EVALUATION

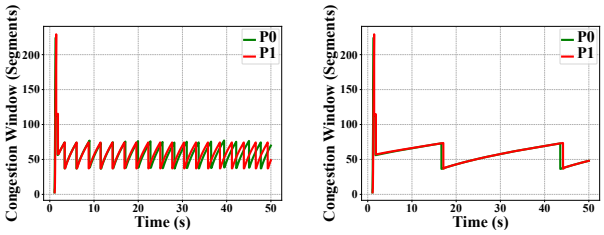
To demonstrate the behavior of the MPQUIC implementation in ns-3, a set of use cases are presented. We first show the scalability of our implementation. Then, we illustrate the performance with different congestion control algorithms and schedulers. Figure 6 presents the network topologies used in our evaluation and Table 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.

Table 1: Path Settings

Setting	Bandwidth	One-way delay	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%

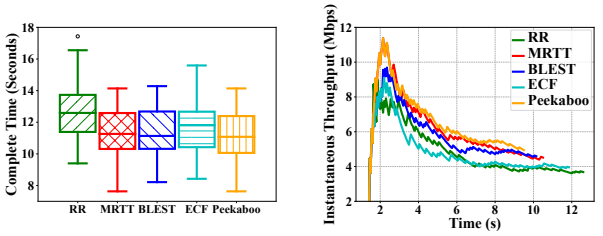
4.1 Scalability of Multiple Paths

The scalability of our implementation is demonstrated with the topology of four paths between client and server (Figure 6a), in which each path has the same range of data rate and one-way delay under Setting 0 as listed in Table 1. Figure 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. Figure 7b depicts



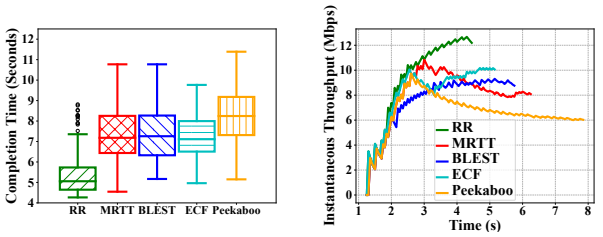
(a) NewReno (b) OLIA

Figure 8: Congestion Window



(a) Completion Time (b) Instantaneous Throughput

Figure 9: Performance Comparison with Different Schedulers in a Stable Scenario



(a) Completion Time (b) Instantaneous Throughput

Figure 10: Performance Comparison with Different Schedulers in an Unstable Scenario

the instantaneous throughput for several paths. The performance is significantly improved with four paths compared with the single-path transmission.

4.2 Performance of Different Congestion Control Algorithms

The evaluation of the congestion control algorithms uses the topology shown in Figure 6b and explores under Setting 1 on each path. RR is the default scheduler in this evaluation. Figure 8 presents the congestion window of NewReno and OLIA by transmitting unlimited data in 50 seconds, where the segment size is 1460 bytes.

4.3 Flexibility with 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 stable and unstable scenarios under the two-path topology (Figure 6b). In a stable scenario, P0 uses Setting 0 and P1 uses Setting 1, indicating that P1 should always have a better performance than P0 with higher bandwidth and lower delay. Figure 9a and 9b depict the completion time and the instantaneous throughput of transmitting 5 MB data with different schedulers for 50 rounds in the stable scenario. Figure 10a and 10b depict those performances in an unstable scenario in that P0 uses Setting 2 and P1 uses Setting 3, indicating P0 has higher bandwidth and higher delay while P1 has lower bandwidth and lower delay.

To present the detailed performance of each path, we also depict the received bytes of each path. Figure 11 presents the received bytes for different schedulers in the stable scenario. Figure 12 also presents the stable 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 S1 uses Setting 0. By doing so, we are able to explore the adaptivity of different schedulers. Furthermore, Figure 13 explores the performance of different schedulers in the unstable scenario.

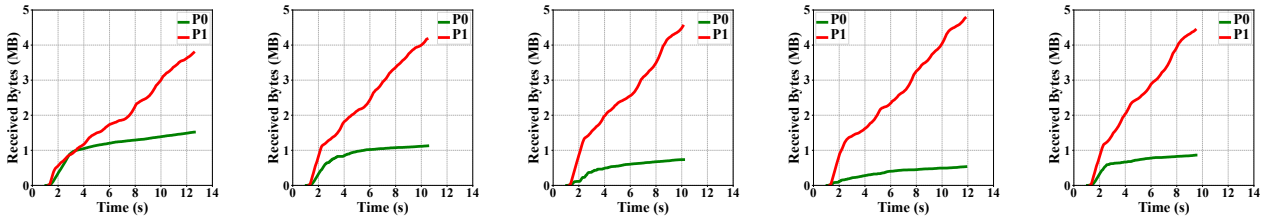
5 CONCLUSIONS

In this paper, 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. The performance of scalability with more paths and interoperability to connect to external emulation interfaces will be evaluated in our future work. Exploring better scheduling and congestion control algorithms is also considered in the future.

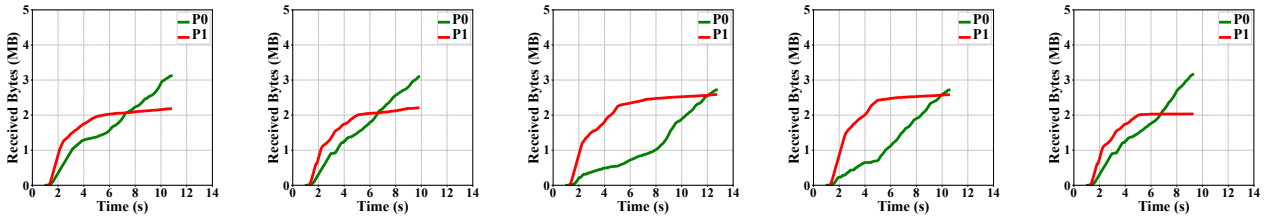
REFERENCES

[1] Stuart Cheshire, David Schinazi, and Christoph Paasch. 2017. Advances in Networking, Part 1. https://developer.apple.com/videos/play/wwdc2017/707/
[2] Alvise De Biasio, Federico Chiariotti, Michele Polese, Andrea Zanella, and Michele Zorzi. 2019. A QUIC Implementation for ns-3. In *Proceedings of the 2019 Workshop on ns-3 (WNS3 2019)*. Florence, Italy, 1–8.
[3] Quentin De Coninck and Olivier Bonaventure. 2017. Multipath QUIC: Design and Evaluation. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT 2017)*. Seoul/Incheon, South Korea, 160–166.
[4] Quentin De Coninck and Olivier Bonaventure. 2021. Multiflow QUIC: A Generic Multipath Transport Protocol. *IEEE Communications Magazine* 59, 5 (2021), 108–113.
[5] Simone Ferlin, Özgü Alay, Olivier Mehani, and Roksana Boreli. 2016. BLEST: Blocking Estimation-based MPTCP Scheduler for Heterogeneous Networks. In *Proceedings of the 2016 IFIP Networking Conference and Workshops (IFIP Networking 2016)*. Vienna, Austria, 431–439.
[6] Sangtae Ha, Injong Rhee, and Lisong Xu. 2008. CUBIC: A New TCP-friendly High-speed TCP Variant. *SIGOPS Oper. Syst. Rev.* 42, 5 (2008), 64–74.
[7] Jana Iyengar and Martin Thomson. 2021. QUIC: A UDP-based Multiplexed and Secure Transport. RFC 9000. https://doi.org/10.17487/RFC9000
[8] Ramin Khalili, Nicolas Gast, Miroslav Popovic, and Jean-Yves Le Boudec. 2013. MPTCP is not Pareto-optimal: Performance Issues and a Possible Solution. *IEEE/ACM Transactions on Networking* 21, 5 (2013), 1651–1665.



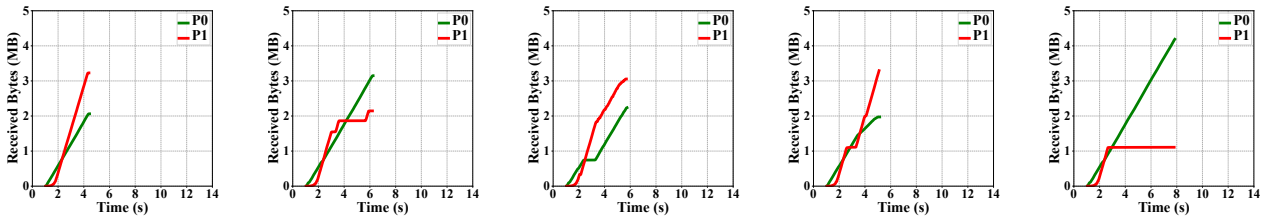
(a) RR (b) MRTT (c) BLEST (d) ECF (e) Peekaboo

Figure 11: Received Bytes of Two Paths under the Stable Scenario



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

Figure 12: Received Bytes of Two Paths under the Stable Scenario with Swapped Setting after 5 Seconds



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

Figure 13: Received Bytes of Two Paths under the Unstable Scenario

[9] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. 2017. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM 2017)*. Los Angeles, USA, 183–196.

[10] Yeon Lim, Erich M. Nahum, Don Towsley, and Richard J. Gibbens. 2017. ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths. In *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies (CoNEXT 2017)*. Seoul/Incheon, South Korea.

[11] Yanmei Liu, Yunfei Ma, Quentin De Coninck, Olivier Bonaventure, Christian Huitema, and Mirja Kühlewind. 2023. *Multipath Extension for QUIC*. Internet-Draft draft-ietf-quic-multipath-04. Internet Engineering Task Force. https://datatracker.ietf.org/doc/draft-ietf-quic-multipath/04/

[12] Christoph Paasch, Gregory Detal, Fabien Duchene, Costin Raiciu, and Olivier Bonaventure. 2012. Exploring Mobile/WiFi Handover with Multipath TCP. In *Proceedings of the 2012 ACM SIGCOMM Workshop on Cellular Networks: Operations, Challenges, and Future Design (CellNet 2012)*. Helsinki, Finland, 31–36.

[13] Umberto Paro, Federico Chiariotti, Anay Ajit Deshpande, Michele Polese, Andrea Zanella, and Michele Zorzi. 2020. 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 2020)*. Alicante, Spain, 19–26.

[14] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. 2011. Improving Datacenter Performance and Robustness with Multipath TCP. In *Proceedings of the ACM SIGCOMM 2011 Conference (SIGCOMM 2011)*. Toronto, Canada, 266–277.

[15] Dipankar Raychaudhuri, Kiran Nagaraja, and Arun Venkataramani. 2012. MobilityFirst: A Robust and Trustworthy Mobility-centric Architecture for the Future Internet. *SIGMOBILE Mob. Comput. Commun. Rev.* 16, 3 (2012), 2–13.

[16] Michael Scharf and Sebastian Kiesel. 2006. Head-of-line Blocking in TCP and SCTP: Analysis and Measurements. In *Proceedings of 2006 IEEE Global Communications Conference (GLOBECOM 2022)*. San Francisco, USA, 1–5.

[17] Xiang Shi, Lin Wang, Fa Zhang, Biyu Zhou, and Zhiyong Liu. 2020. PStream: Priority-based Stream Scheduling for Heterogeneous Paths in Multipath-QUIC. In *Proceedings of the 29th International Conference on Computer Communications and Networks (ICCCN 2020)*. Honolulu, USA, 1–8.

[18] Tobias Viernickel, Alexander Froemmgen, Amr Rizk, Boris Koldehofe, and Ralf Steinmetz. 2018. Multipath QUIC: A Deployable Multipath Transport Protocol. In *Proceedings of 2018 IEEE International Conference on Communications (ICC 2018)*. Kansas City, USA, 1–7.

[19] Damon Wischik, Costin Raiciu, Adam Greenhalgh, and Mark Handley. 2011. Design, Implementation and Evaluation of Congestion Control for Multipath TCP. In *Proceedings of the 8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 2011)*. Boston, USA, 99–112.

[20] Hongjia Wu, Özgü Alay, Anna Brunstrom, Simone Ferlin, and Giuseppe Caso. 2020. Peekaboo: Learning-based Multipath Scheduling for Dynamic Heterogeneous Environments. *IEEE Journal on Selected Areas in Communications* (2020).

[21] Wenjun Yang, Lin Cai, Shengjie Shu, and Jianping Pan. 2022. Scheduler Design for Mobility-aware Multipath QUIC. In *Proceedings of 2022 IEEE Global Communications Conference (GLOBECOM 2022)*. Rio de Janeiro, Brazil, 2849–2854.

[22] Wenjun Yang, Shengjie Shu, Lin Cai, and Jianping Pan. 2021. MM-QUIC: A Mobility-aware Multipath QUIC for Satellite Networks. In *Proceedings of the 17th International Conference on Mobility, Sensing and Networking (MSN 2021)*. Exeter, United Kingdom, 608–615.