On the Latency of Multipath-QUIC in Real-time Applications

Vu Anh Vu
Institute of Communications Technology
Leibniz University Hannover
vuanh.vu@ikt.uni-hannover.de

Brenton Walker
Institute of Communications Technology
Leibniz University Hannover
brenton.walker@ikt.uni-hannover.de

***Abstract*—Recently, the ubiquity of broadband wireless networks has encouraged the growth of real-time network applications. Their strict latency requirements present a challenge for traditional Internet protocols which tend to be designed to optimize mean performance. Two recent developments that show promise for addressing the challenges of real-time applications are QUIC, a UDP-based protocol that has many features of TCP, and multipath transmission, an extension allowing transport layer protocols to transmit data simultaneously over multiple network paths and interfaces. In this paper, we present MAppLE (MPQUIC Application Latency Evaluation platform), which provides the instrumentation needed to evaluate and develop MPQUIC stream multiplexers, stream schedulers, and multipath packet schedulers. We also present our `NineTails` scheduler, a multipath MPQUIC scheduler that utilizes selective multipath redundancy to control tail loss and near-tail loss latencies. Our experimental results show that in a lossy asymmetric heterogeneous wireless network, our proposed scheduler reduces outlier latencies compared to other existing scheduling algorithms, and improves Quality of Experience (QoE) in video streaming by inducing fewer playback rebuffering events.**

***Index Terms*—DASH, video streaming, multipath scheduling, scheduler, multiplexer, multi-stream, transport layer protocol, tail loss, application-level latency, evaluation platform**

I. INTRODUCTION

Real-time network applications such as media streaming, Virtual/Augmented Reality, Tele-Operated Driving, and Remote Surgery, have demanding Quality of Service (QoS) requirements. For example, Tele-Operated Driving requires information exchange between user equipment (UE) and a V2X Application Server with a data rate of *25Mbps*, the latency of *5ms*, and reliability of *99.999%* [1]. These requirements are difficult to meet using conventional protocols such as TCP. Newer protocols designed for lower latency, even for short flows, and multipath redundancy offer new possibilities for meeting the requirements of real-time applications.

QUIC [2] is a general-purpose transport-layer network protocol that supports multiple streams and a 0-RTT/1-RTT handshake mechanism. It offers reliable connections with flow and congestion control, similar to TCP, but unlike TCP, QUIC improves support for real-time applications with instant handshakes and faster loss detection and recovery [3]. QUIC is supported by the Google Chrome web browser and YouTube video streaming service. By 2017, QUIC was estimated to account for *7%* of the world's total Internet traffic. However Bhat et al. [4] found that QUIC does not improve the performance of DASH video streaming, and actually leads to degradation in some aspects. Therefore, further research and development are needed to make QUIC an acceptable transport protocol for real-time applications.

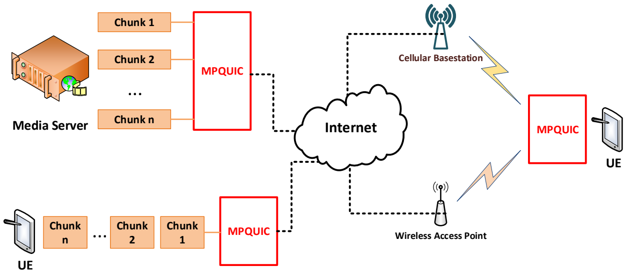


Fig. 1. Video-on-demand and peer-to-peer live streaming using MPQUIC with mobile cellular and WiFi networks simultaneously

Nowadays heterogeneous wireless networks have become ubiquitous, with most UEs having multiple wireless interfaces (802.11x WiFi, cellular, mmWave, VLC). Several transport protocols have recently been extended to support multiple network paths while appearing as a single logical connection to the application layer, for example, CMT-SCTP [5], MPTCP [6], and MPQUIC [7, 8]. Multipath redundancy is one popular and promising solution for controlling data delivery latency, particularly in mobile and wireless-capable devices. Multipath protocols have some drawbacks, however, such as Head-of-Line (HoL) blocking and poor path scheduling. In this area, MPQUIC can offer more with stream-aware scheduling [9] and stream multiplexing.

Figure 1 illustrates scenarios where MPQUIC could be used to boost video streaming performance. On-demand video servers can benefit from QUIC multi-streaming by queueing their stored video chunks into multiple streams. If one stream is blocked due to HoL-blocking or packet loss, the other streams can still use the available link capacity, thus increasing bandwidth utilization. In live video streaming between UEs, where video chunks are continuously produced, MPQUIC can aggregate throughput and reduce transmission interruptions to provide smoother video playback.

MPQUIC has many features that should help it support real-time applications. However, because QUIC is a new protocol and has not been standardized, it lacks a unified platform to evaluate its latency performance. Well-known full-featured traffic generators such as D-ITG [10] do not support QUIC. qperf [11] is a QUIC-compatible greedy payload generator, but lacks many features of a traffic generator such as arriving rate control and end-to-end latency logging. Video streaming measurements with QUIC have been done in [4], but the measuring setup is not portable to other experiments.

Wireless networks are prone to packet loss due to radio signal fading, shadowing, and interference. Reliable transport protocols, such as TCP and QUIC, have two main methods for recovering lost data segments. (1) Fast retransmit of a lost segment is triggered when the sender receives a number of duplicate acknowledgments (ACKs), which could be a constant value or be adaptive as in Early Retransmission [12]. (2) When a segment is lost, but the trigger conditions are not met, the protocol waits for the Retransmission Timeout (RTO) before it retransmits the segment. This usually occurs when the lost segments are the end of the flow (tail loss). To avoid this case, [13] proposes a Tail Loss Probe algorithm to recover more quickly from tail losses. However, the timeout delay can be as long as $2 * RTT$. Another approach to mitigating packet losses is to transmit multiple copies of data segments (multipath redundancy); if only one copy is lost, the sender does not need to retransmit it. This idea works quite well in combination with multipath transmissions with multiple copies of segments sent in multiple paths and has been used in several MPTCP schedulers [14, 15, 16, 17]. However, MPTCP does not know which segment is at the tail; therefore, it cannot reliably guarantee tailed segment duplication to avoid tail loss. MPQUIC with the stream-aware feature has the potential to solve this issue.

In this paper, we introduce a unified **MPQUIC** **App**lication **L**atency **E**valuation platform (**MAppLE**) to evaluate and develop MPQUIC with modular multiplexers, stream schedulers, and packet schedulers. We present the **`NineTails`** multipath scheduler - a stream-aware scheduler that reduces tail loss or near-tail loss latencies by selectively transmitting duplicate data at the end of a stream. Our results show reductions in latency outliers compared to other multipath schedulers. **`NineTails`** also induces smaller rebuffering frequency when evaluated in DASH video streaming.

II. APPLICATION LATENCY IN MPQUIC

Application end-to-end latency is one of the most important metrics for real-time applications. It is the time between when the source application sends a unit of data (**a message**) to its network socket(s), to the moment the destination application receives the message from its corresponding network socket(s). There are many potential sources of delay, which can be addressed in different ways [18]. In this paper, we focus on delays originating at the transport layer.

Fig. 2 shows a histogram of latencies of *40kB* messages transmitted by MPQUIC over the emulated network described in section V. The latencies have a minimum value of $\approx 70ms$, consistent with an OWD of $\approx 50ms$ and the transmission delay. However, a large fraction of the messages arrives much later for a variety of reasons. We will elaborate on the factors that can increase end-to-end latency in the following sections.

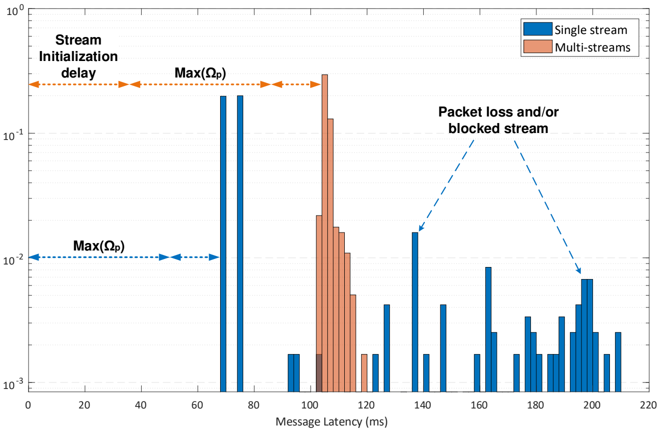


Fig. 2. Message latency histogram comparison between Single-stream and Multi-stream with RoundRobin stream scheduler.

A. HoL Blocking

In order to be network-friendly, QUIC has flow control and congestion controls that keep the bytes in flight below $\min(receive\_window, congestion\_window)$, accordingly control the data transmission rate. Head-of-Line Blocking (HoL) occurs when the congestion window, which indicates the network capacity, still has enough space to send more data, but the sliding window does not allow it to send further until some preceding segments (potentially lost) are ACKed. This blocks the data flow until it receives the ACK for the lost segment, which could take at least $2RTT$ after the segment was first transmitted, plus reordering time. Multimedia applications using TCP can mitigate Head-of-Line Blocking by opening multiple parallel TCP connections [19], but this must be implemented in the application.

HoL blocking occurs even more frequently in multipath transmission in networks with asymmetric paths, where the RTTs and capacities of the paths are very different. In this case, segments often arrive out of order due to the difference in the paths' delays. The paths with shorter delays might need to wait for the segments transmitted by the high-latency paths to arrive (or worse, to be retransmitted). MPTCP's original solution for this issue is Opportunistic Retransmission [14], which allows a path with a free congestion window to retransmit segments already sent on other paths. However, this solution causes a lot of unnecessary redundancy in light traffic and does not have any effects with greedy sources. Ferlin et al. [20] proposed a blocking estimation scheduler to minimize HoL blocking and increase goodput.

MPQUIC mitigates HoL Blocking by using multiplexed streams. Messages can be queued on multiple streams, so if one stream is blocked, the others can still transmit their data. This is similar to SCTP's multi-streaming, but QUIC has more flexible stream establishment, as well as better handling of packet loss due to its two-level numbering [21].

B. Loss Recovery

QUIC's loss detection and recovery mechanisms are similar to those of TCP, but with a few differences. QUIC avoids TCP's "retransmission ambiguity" by separating the packet numbers from the data sequence number. QUIC's packet number increases monotonically and implies transmission order. When a packet is detected lost, QUIC encapsulates the lost data frame in a new packet with a new packet number; hence it knows exactly which packet is ACKed when receiving an ACK. This design offers more accurate RTT measurements and improves loss detection.

With Fast Retransmission, QUIC takes at least $\frac{3}{2}$RTT to correct a lost segment. MPQUIC has the ability to further reduce loss recovery time by retransmitting segments lost on a high-latency path over a lower-latency path. This reduces the loss recovery time to $\mathrm{RTT}_{losspath} + \min(\mathrm{OWD}_i)$ However, in some cases, a loss event will not be detected by Fast Retransmission.

- Tail Loss: Packet loss at the end of a flow.
- Large mid-flow loss: Loss of an entire window of data or ACKs during transmission.
- Fast Retrans. duplicated ACK threshold is too high.
- Long RTT that exceeds Retransmission Timeout (RTO).

In these cases, the lost packet must wait for the RTO (200ms in the MPQUIC implementation) to be retransmitted. Meanwhile, the whole stream is blocked. Tail Loss is the most common of these cases because it can be caused by a single lost packet. Tail Loss Probe [13] mitigates this issue and reduces the blocking time to less than the RTO, typically to $\max(2 \times RTT, 10ms)$ if there are multiple outstanding packets, or $\max(2 \times RTT, 1.5 \times RTT + RTO/2)$ if there is only one. In any case, the blocking time is non-trivial and contributes significantly to the latency outliers.

C. *Packet scheduling*

In MPQUIC the packet scheduling algorithm decides how to use the available paths most efficiently. For example, packet scheduling can be used to actively reduce loss recovery time by injecting redundant data into paths. The data can be identical (such as in the **`Redundant`** schedulers [14]) or encoded (forward error correction). However, it is not always advantageous to transmit a message on all paths. If one path has high OWD and low capacity, transmitting even a small piece of data on it may delay message completion.

D. *Multi-streaming and Stream scheduling*

QUIC multi-streaming can help mitigate flow blocking and under-utilized bandwidth, but it does not necessarily reduce message latency. For example, establishing a stream for each message would eventually increase the average waiting time of messages in the stream. Moreover, since each QUIC stream has its own receive window, it is possible that an "initialization delay", which is the time for the receive window to scale, may occur. This stream initialization delay in MPQUIC is approximately the largest RTT of the paths. Fig. 2 compares the message latency of multi-streaming to single-stream. In the single-stream case, only the first message suffers from an increased initialization delay. Therefore the minimum and average latencies are lower then multi-streaming, where all messages have an additional delay of $\approx 50ms$. On the other hand, sometimes, the single stream is blocked by HoL or packet loss, adding extra latency.

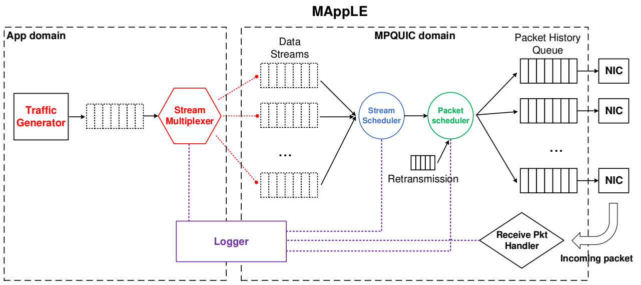


Fig. 3. MPQUIC Application Latency Evaluation (MAppLE) platform

III. THE MAPPLE EVALUATION PLATFORM

We have developed the modular **M**PQUIC **App**lication **L**atency **E**valuation (**MAppLE**) platform, which provides the instrumentation and modularity needed to design and evaluate scheduling and multiplexing algorithms in MPQUIC. The platform includes the core MPQUIC components, along with a pluggable multiplexer, stream scheduler, and packet scheduler modules. Fig. 3 illustrates the architecture of MAppLE. The core element of the platform is the MPQUIC implementation [7], with our modifications to enable modular schedulers and multiplexers. This design respects the IETF's QUIC specification with support for multipath transmission. The components in the application domain are out of MPQUIC's scope. The primary components that we contribute are:

- **Traffic Generator** generates payloads in the form of messages - pieces of pseudo data, tagged with an ID that can be tracked at the receiver. This allows us to measure per-message latency statistics. The generated inter-arrival times and message sizes can be configured from a list of common random distributions (Constant, Uniform, Exponential, Pareto, Poisson, etc.). Generated messages are placed into an intermediate queue. This queue has two purposes: first, it allows the Traffic generator to mimic the application's blocked/non-blocked socket options. Second, it emulates the TCP send queue, facilitating comparisons with MPTCP.
- **Stream Multiplexer** applies multiplexing policies, such as single stream and multiple parallel streams. More complex policies can be easily defined.
- **Stream Scheduler** selects which stream's data to transmit. Weighted-RoundRobin is a general-purpose scheduler that works well. Given specific requirements and use cases, more tailored schedulers can also be defined.
- **Multipath Scheduler** encapsulates the data given by the stream scheduler and distributes it amongst the available network paths. The Lowest-RTT-path first (**`LowRTT`**) and **`RoundRobin`** are two simple general-purpose multipath scheduling algorithms that offer fair all-around performance in many situations.

One advantage of QUIC, compared to other transport layer protocols, is that it is implemented in user-space. Therefore, QUIC can share cross-layer information with applications without any special API. The MAppLE platform establishes a shared channel where its components exchange their state and cooperate with each other. Instead of executing their algorithms independently, schedulers can work together to create more sophisticated strategies.

In order to measure latencies at the message, frame, and packet level, data is time-stamped at each encapsulation level of both the sender and receiver. Clocks are synchronized via the Precision Time Protocol (PTP).

IV. THE NINETAILS MULTIPATH SCHEDULER

Based on our MAppLE evaluation and development platform, we built a multipath packet scheduler that aims to reduce message latency outliers in lossy heterogeneous wireless networks with asymmetric paths.

The majority of current consumer mobile devices have two wireless interfaces: one mobile cellular (3G, LTE) interface, which provides decent throughput and latency, and one WiFi (802.11x) interface, which provides relatively lower latency and smaller throughput. Both of them can be lossy due to radio interference or overflow in middle-boxes. ACK-based loss recovery needs at least 1 RTT, which could be as high as hundreds of milliseconds, to recover a lost data segment. Furthermore, a lost segment can block the path from sending data, degrade its throughput, and increase the waiting time of data in higher-level queues at both senders and receivers. The result is long-tailed latency probability distributions. For real-time applications, such high latencies are unacceptable.

The premise of multipath redundancy is to replicate data segments over multiple paths so that if one replica is lost, the other can still arrive. Thus, it reduces tail latency and path blocking. Redundancy is a proactive loss recovery approach, in that it can correct a loss even before it is detected. Many redundant multipath variants have been proposed for MPTCP, however, due to design differences, redundant schedulers for MPQUIC must be implemented differently.

A. Multipath Redundant Scheduling in MPQUIC

Unlike MPTCP, where each connection has a single queue of undelivered segments, in MPQUIC there are potentially multiple streams being fed by the application. The data from the streams is encapsulated into data frames by the stream scheduler according to its own priorities. There is no single chronology of data frames. To illustrate the differences this causes, consider the implementation of the **Redundant** scheduler. In MPTCP's implementation, when space is available in the congestion window of a path, the **Redundant** scheduler queues the oldest un-ACKed segment that has not yet been sent on that path. In this way, almost every segment is sent on every path. In MPQUIC the notion of "oldest" data is not well defined, and not easily accessible to the path scheduler. However, since there are multiple paths, at any given time there is always a **leading path** that has transmitted the most recent data frame. The **trailing path(s)** will most likely transmit redundant frames already sent by the leading path. Instead of looking for data to transmit from a shared queue, the scheduler must identify the leading path and check its packet history to find the oldest outstanding data frame that has not yet been duplicated on a trailing path.

We have implemented the **Redundant** packet scheduler in MPQUIC. As with many redundant schedulers, this scheduler provides lower goodput than the default **LowRTT** scheduler, but for lightly-loaded applications, it offers the best control over packet latency outliers. However, it is not necessarily the best at controlling latency outliers at the application level.

Algorithm 1 **NineTails** Scheduling Algorithm

```
function NineTailsScheduler(availablePathsList, highestRatePath, lowestRTTPath)
    highestRate ← highestRatePath.rate
    ▷ Select leading and redundant paths
    bool shouldRedundant ← messageRemainData/highestRate < min(3*highestRate/2, minRTOTimeout + highestRate/2)
    if shouldRedundant then
        selectedPath ← highestRatePath
        redundantPathList ← availablePathsList - selectedPath
    else
        selectedPath ← lowestRTTPath
    ▷ Select Data Frame to transmit
    newFrame ← StreamScheduler.SelectStreamFrame()
    newPacket ← packPacket(newFrame)
    sendPacket(newPacket, selectedPath)
    for all redundantPath ∈ redundantPathList do
        duplicatedFrame ← findUnsentFrame(selectedPath.PacketHistory)
        dupPacket ← packPacket(duplicatedFrame)
        sendPacket(dupPacket, redundantPath)
```

B. Near-tail Redundancy

The problem with the plain **Redundant** algorithm is that, while it reduces latencies at the packet and frame level, its reduced throughput increases the transmission time of the whole message. We can best reduce message latency by being selective about when to employ redundancy.

When a frame in the middle of a message is lost, the succeeding frames can still be transmitted until the sliding windows are full, congestion windows are full, or fast-retransmit is triggered. Generally, the lost frame will only cause negligible extra message latency as long as it is recovered before the recipient of the last frame of the message. On the other hand, when the loss occurs in the last few frames of the message it causes significant extra latency on the order of the RTO. Moreover, when a loss occurs near the end of the message, it does not matter how high the aggregated throughput is, because all the received frames have to wait for the lost one so they can be delivered in order.

We propose the **NineTails** scheduler, named after the mythological nine-tailed fox with one head and multiple tails, based on the idea of only using redundancy near the ends of messages. Through most of a message, **NineTails** selects the available path with the lowest RTT to transmit each new frame, just like **LowRTT**. When it gets near the tail of the message, **NineTails** switches to REDUNDANT_MODE, transmitting the frames via all available paths. In this way, it benefits from aggregated throughput through the bulk of the message, while still avoid additional latency caused by near tail loss recovery and tail loss RTO. REDUNDANT_MODE is triggered by looking at the length of the remaining data in the message, estimating the time needed to transmit it with

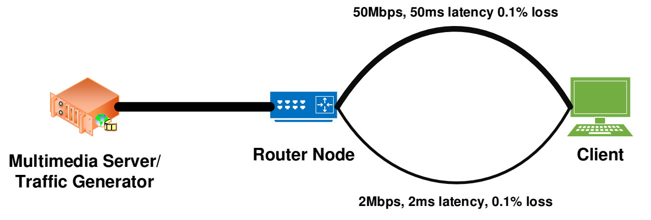


Fig. 4. Emulated network testbed with two paths

full redundancy, and comparing it to the additional recovery time non-redundant transmission would suffer if the frame were lost. The scheduler switches to REDUNDANT_MODE when the loss recovery time is greater than the transmission time with full redundancy.

$$\frac{messageRemainData}{highestRate} < \min\left(\frac{3}{2} * highestRate, minRTOTimeout + \frac{1}{2} * highestRate\right) \tag{1}$$

This concept utilizes the stream-aware feature of QUIC to get the $messageRemainData$. Other path statistic parameters are collected during the transmission.

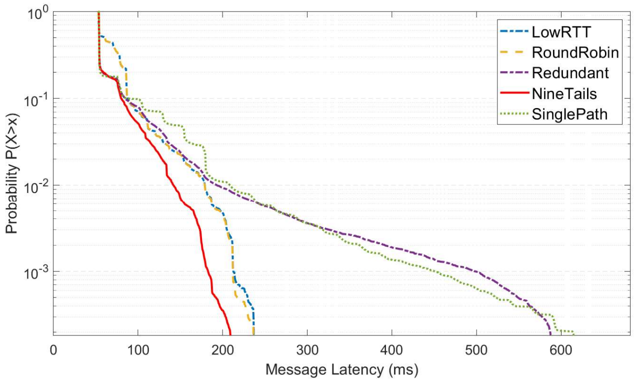
In our implementation, due to the fluctuation in the estimated data sending rate, we repeat the tail determination at every path selection step. Algorithm 1 describes our algorithm for selecting redundant paths and data frames.

V. EVALUATION

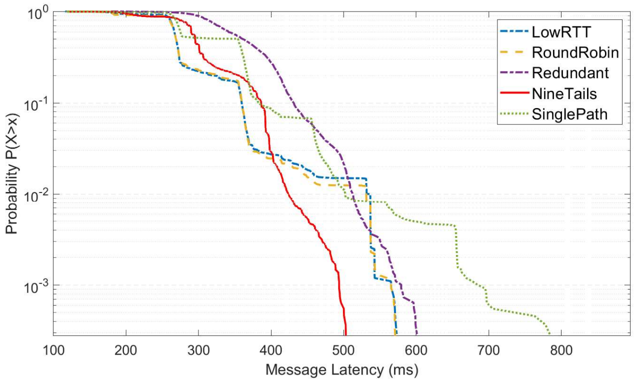
A. Setup

The proposed schedulers are evaluated in an Emulab [22] emulated network. Fig.4 illustrates the evaluated topology with two MoonGen-based [23] emulated links representing cellular and WiFi paths. The ”Mobile network” path with a capacity of 50Mbps, delay of 50 ms, and a Bernoulli packet loss rate of 0.1%, taken from an LTE measurement in [24]. The ”WiFi” path emulates an IEEE 802.11p link in vehicular networks, which is defined to have different data rates between 3 and 27 Mbps. However, considering that the bandwidth is shared with road safety applications and control signals, and the practical throughput can be influenced by distance and fading, we chose emulated link with 2 Mbps capacity, 2 ms delay, and 0.1% loss rate (the actual end-to-end loss rate could be higher due to congestion). We used the uncoupled CUBIC congestion control algorithm, which means each path has an independent congestion window. The reason we did not use multipath coupled congestion control, such as OLIA and BALIA, is that with them, the congestion window of the higher latency lossy path would shrink over time, which would make our experimental results inconsistent and dependent on the length of each experiment.

The first experiment evaluated the schedulers within our MAppLE platform implementation [25] with a generated traffic rate of 6.4 Mbps, which is enough to get MPQUIC to use both paths, but does not congest both of them at a same time. We conducted experiments using several message sizes and inter-arrival times with the targeted rate (*20kB-25ms, 40kB-50ms, 80kB-100ms, 160kB-200ms, 200kB-250ms, 400kB-500ms, 800kB-1000ms*). Each of them was evaluated over 2 minutes × 50 runs. Our experimental metrics are the q-quantiles at 99% and 99.9% of the message latencies. We compared our proposed scheduler with the **LowRTT**, **RoundRobin**, and **Redundant** schedulers, as well as **SinglePath**.



(a) Message size 20kB



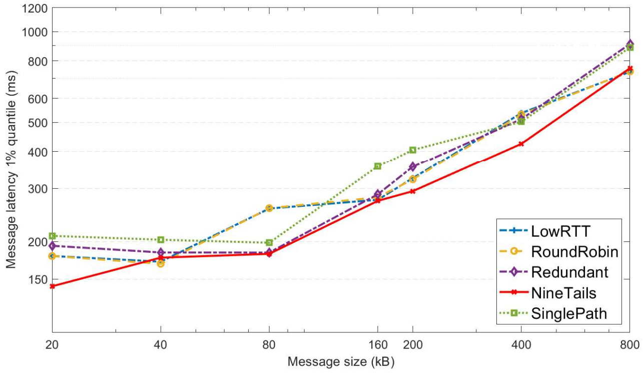
(b) Message size 400kB

Fig. 5. Message latency CCDF comparison between schedulers with small message size (*20kB*) and large message size (*400kB*) on a log scale

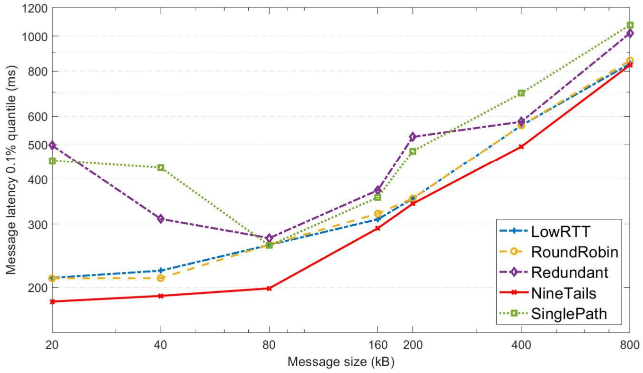
In the second experiments, we used a MPQUIC-compatible DASH video streaming application to stream a 1-minute 4k video (*3840 × 2160*) with a bitrate of *6592kbps* (50 runs). Adaptive bitrate streaming was disabled to avoid the transmission rate variation. The streamed video segment duration is 200ms, and the maximum client playback buffer size is two segments (400ms). We use this relatively small segment and playback buffer size to see how the packet loss causing retransmission latency of 500ms-1s affects the QoE in delay-sensitive applications. Our performance metric is video rebuffering frequency, which is an important metric for measuring video playback QoE [26].

B. Message latency with generated traffic

Fig 5(a) shows the Complementary Cumulative Distribution Function (CCDF) of small message (*20kB*) latency. In this case, the redundant-type schedulers are expected to have lower latency overall, since the latency reduction offered by bandwidth aggregation is less than the loss recovery time. However, in the result we observed longer tail distribution



(a) Message latency at 1% quantiles



(b) Message latency at 0.1% quantiles

Fig. 6. Quantiles of message latency with multiple message sizes

at the **Redundant** scheduler, due to its lower aggregate throughput compared to the other schedulers'. When a packet loss occurs on the higher bandwidth path, the throughput drops below the generated traffic rate, increasing the waiting period of messages in the sender queue. This circumstance usually happens when the paths have asymmetric bandwidth, which leads to one path transmitting old data frames that have been sent, but not ACKed, by the other subflow [16]. Therefore, the lower-bandwidth path cannot make up for the throughput drop on the leading path. The similar tail distributions of **SinglePath** and the **Redundant** scheduler supports this explanation because their total throughputs are both equal to the leading path's throughput. Most of the time, **NineTails** behaves identically to the **Redundant**. However, when the throughput drops, the amount of queued data in the stream increases and triggers **NineTails** to disable redundancy. For that reason, **NineTails** has the lowest latency at the tail of the distribution.

With large message sizes (*400kB*), the schedulers with a higher aggregate throughput gain a greater latency reduction, whereas the tail and near-tail loss recovery time remains the same. Therefore, the **Redundant** scheduler has the highest latency overall. Fig. 5(b) shows that **NineTails** benefits from having higher aggregate throughput at the beginning of the message transmission and is still able to avoid the latency penalties of the tail and near-tail loss at the end of it. Hence it has an average latency in between the others and much-reduced latency at after the 99th percentiles.

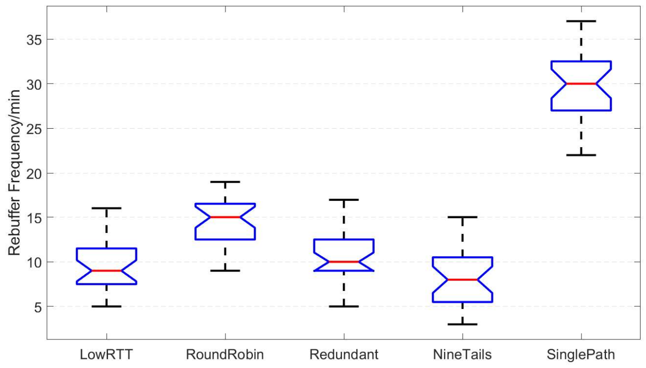


Fig. 7. Video playback rebuffering frequency

Fig. 6 shows the message latency at the 1% and 0.1% quantiles of all message sizes. We can see that **NineTails**, in general, has the lowest message latencies at both quantiles. The latency reductions at the 1% quantile are not drastic. The latencies at 0.1% quantile, however, highlight **NineTails**'s improvement in the tail and near-tail loss recovery, cutting down the outlier latencies with an advantage of up to 150ms.

*C. The **NineTails** scheduler in video streaming*

In experiments with a video streaming data source, Fig. 7 shows the average rebuffering frequency of each scheduler. **NineTails** has slightly lower (2-3 rebuffering events/min) rebuffering frequency than the **LowRTT** and **Redundant**, and only half and one third as many rebuffering events, respectively, compared to **RoundRobin** and **SinglePath**.

The results tend to favor the redundant schedulers because the client's playback buffer is tiny. Although the **NineTails** scheduler does not consider messages' deadline, it has better results in this experiment, because the maximum deadline for the video segments (2 segments, $\approx$ 400ms) and the video chunks are quite small, which is to the advantage of the redundant schedulers. Also, the video bitrate is low enough to avoid overloading the total transmission rate (anyhow, in adaptive streaming, bitrate would be adjusted to the transmission rate), so that a lower aggregated rates is not a big disadvantage to the redundant schedulers. If the buffer size is larger, the client can preload more frames to avoid rebuffering, and a scheduler providing a higher goodput, such as the **LowRTT**, would have more benefits.

VI. DISCUSSION

While evaluating application traffic on our platform, we observed some issues that are worth investigating. The first problem is scheduling a real-time message at both packet-level and stream-level with a time deadline. As a transport layer protocol, (MP)QUIC need not be aware of the deadline of its application data. However, in many cases, fulfilling the deadline is even more important than reducing average latency or latency outliers. For example, in our video streaming experiment in section V, the client's maximum buffer size is two video segments ($2 * 200ms$). Rebuffering event will definitely occur if a segment is not delivered within the maximum time of 400ms. In this case, an important metric is the quantile of latency below 400ms $P(L_m \leq 400ms)$. This metric has been taken into account in [17]; however, only packet-level deadlines in MPTCP were investigated.

The second issue, which we mentioned in section II, is whether MPQUIC should use multiple streams for consecutively arriving data from applications. The trade-off in Fig. 2 raises the questions: (1) What is the optimal number of concurrent streams? And (2) which stream should we put the messages to? Too many non-priority streams would drastically increase messages' wait duration the stream queues. The stream initialization delay is also an important factor, especially with multipath. This delay can be avoided with a stream scheduling algorithm. Also, multi-streaming only offers a benefit when the streams are blocked frequently, which could be avoided or worsened by the packet scheduler.

VII. CONCLUSION

In this paper, we introduced MAppLE - an MPQUIC Application Latency Evaluation platform providing the instrumentation to evaluate and build performance-critical MPQUIC components: stream multiplexers, stream schedulers, and multipath schedulers. MAppLE enables us to develop those components with many types of network application traffic without the burden of implementing and deploying the protocol (and its components) into the applications.

The **`NineTails`** scheduler was built on MAppLE with the goal of reducing latency outliers caused by losses near the end and at the end of data messages. It uses multipath scheduling with a selected redundancy algorithm, enjoying the benefits of both redundant and non-redundant schedulers: higher aggregate throughput and quick loss recovery. Therefore, it offers reduced latencies outliers for varied message sizes, by the offset of at least one longest RTT the paths. The video streaming results confirms the QoE improvement when using **`NineTails`**.

There are many open issues in MPQUIC regarding the stream-multiplexing and latency deadline satisfaction to support real-time applications. In the future, we will attempt to address these two issues in combination, to achieve an end-to-end solution for application latency reduction with transport layer multipath transmission.

ACKNOWLEDGMENT

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 227198829 / GRK1931 SocialCars.

REFERENCES

[1] ETSI, “5g service requirements for enhanced v2x scenarios (3gpp ts 22.186 version 15.3.0 release 15).”

[2] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, and et al., “The quic transport protocol: Design and internet-scale deployment,” in *SIGCOMM ’17*. ACM, 2017.

[3] J. Iyengar and I. Swett, “QUIC Loss Detection and Congestion Control,” IETF, Internet-Draft draft-ietf-quic-recovery-24, Nov. 2019, work in Progress.

[4] D. Bhat, A. Rizk, and M. Zink, “Not so quic: A performance study of dash over quic,” in *NOSSDAV’17*. ACM, 2017.

[5] 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*, vol. 14, no. 5, pp. 951–964, Oct 2006.

[6] A. Ford, C. Raiciu, M. J. Handley, O. Bonaventure, and C. Paasch, “TCP Extensions for Multipath Operation with Multiple Addresses,” IETF, Internet-Draft draft-ietf-mptcp-rfc6824bis-18, Jun. 2019, work in Progress.

[7] Q. De Coninck and O. Bonaventure, “Multipath quic: Design and evaluation,” in *CoNEXT ’17*. ACM, 2017.

[8] T. Viernickel, A. Froemmgen, A. Rizk, B. Koldehofe, and R. Steinmetz, “Multipath quic: A deployable multipath transport protocol,” in *IEEE ICC’18*, 2018.

[9] A. Rabitsch, P. Hurtig, and A. Brunstrom, “A stream-aware multipath quic scheduler for heterogeneous paths,” in *EPIQ’18*. ACM, 2018.

[10] S. Avallone, S. Guadagno, D. Emma, A. Pescape, and G. Ventre, “D-itg distributed internet traffic generator,” in *QEST ’04*.

[11] A performance measurement tool for quic similar to iperf. [Online]. Available: https://github.com/rbruenig/qperf

[12] J. Blanton, P. Hurtig, U. Ayesta, K. Avrachenkov, and M. Allman, “Early Retransmit for TCP and Stream Control Transmission Protocol (SCTP),” RFC 5827, Apr. 2010. [Online]. Available: https://rfc-editor.org/rfc/rfc5827.txt

[13] N. Dukkipati, N. Cardwell, Y. Cheng, and M. Mathis, “Tail Loss Probe (TLP): An Algorithm for Fast Recovery of Tail Losses,” IETF, Internet-Draft draft-dukkipati-tcpm-tcp-loss-probe-01, Feb. 2013, work in Progress.

[14] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley, “How hard can it be? designing and implementing a deployable multipath tcp,” in *NSDI’12*. USENIX Association, 2012.

[15] A. Frömmgen, T. Erbshäußer, and R. Steinmetz, “A Programming Model for Application-defined Multipath TCP Scheduling,” no. December, 2017.

[16] B. Walker, V. A. Vu, and M. Fidler, “Multi-headed mptcp schedulers to control latency in long-fat / short-skinny heterogeneous networks,” in *CHANTS ’18*. ACM, 2018.

[17] V. A. Vu and B. Walker, “Redundant multipath-tcp scheduling with desired packet latency,” in *CHANTS ’19*. ACM, 2019.

[18] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I. J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl, “Reducing Internet Latency: A Survey of Techniques and Their Merits,” *IEEE Communications Surveys and Tutorials*, vol. 18, no. 3, pp. 2149–2196, 2016.

[19] T. Nguyen and Sen-Ching S. Cheung, “Multimedia streaming using multiple tcp connections,” in *PCCC ’05*, 2005.

[20] S. Ferlin, Ö. Alay, O. Mehani, and R. Boreli, “Blest: Blocking estimation-based mptcp scheduler for heterogeneous networks,” in *IFIP Networking*, 2016.

[21] A. Joseph, T. Li, Z. He, Y. Cui, and L. Zhang, “A Comparison between SCTP and QUIC,” Internet Engineering Task Force, Internet-Draft draft-joseph-quic-comparison-quic-sctp-00, Mar. 2018, work in Progress.

[22] M. Hibler, R. Ricci, L. Stoller, J. Duerig, S. Guruprasad, T. Stack, K. Webb, and J. Lepreau, “Large-scale virtualization in the emulab network testbed,” in *USENIX ATC’08*, 2008.

[23] Moongen traffic shaping delay node. [Online]. Available: https://github.com/brentondwalker/MoonGen

[24] N. Becker, A. Rizk, and M. Fidler, “A measurement study on the application-level performance of lte,” in *2014 IFIP Networking Conference*, 2014, pp. 1–9.

[25] Mapple - mpquic application latency evaluation platform. [Online]. Available: https://github.com/vuva/mpquic-latency

[26] R. K. P. Mok, E. W. W. Chan, and R. K. C. Chang, “Measuring the quality of experience of http video streaming,” in *IFIP/IEEE International Symposium on Integrated Network Management*, 2011.