A NOVEL FRAMEWORK FOR HTTP/2 OVER MPQUIC: DESIGN AND IMPLEMENTATION

Thanh Trung Nguyen[1,2], *Minh Hai Vu*[1], *Phi Le Nguyen*[1,*], *Kien Nguyen*[3]

Abstract

The growing demand for high-performance and reliable web communication has driven significant advancements in transport protocols. This paper introduces an innovative framework for deploying Hypertext Transfer Protocol Version 2 (HTTP/2) over Multipath QUIC (MPQUIC), designed to enhance data transmission efficiency in heterogeneous networks. By integrating HTTP/2's multiplexing and prioritization capabilities with MPQUIC's ability to utilize multiple network paths concurrently, the framework addresses critical challenges in stream scheduling and protocol compatibility. To validate its practicality, we design and implement several stream schedulers within the framework. Comprehensive experimental evaluations in Mininet-WiFi confirm the framework's effectiveness and provide comparative insights into scheduler performance. This work lays a strong foundation for the future development of high-performance web browsing over MPQUIC.

Index terms

HTTP/2; MPQUIC; framework; stream scheduling.

1. Introduction

Web browsing accounts for a large share of global Internet bandwidth, making website loading speed a key factor in assessing performance. Nowadays, more and more services are provided on the web platform, such as video streaming, social networks, etc. With the continuous enrichment of website functions, the types and sizes of objects in websites are becoming more diverse, making website performance optimization challenging. Modern web pages contain numerous elements that must be fetched to load the page fully. To enhance this process, HTTP/2 [1], [2] introduces advanced capabilities such as multiplexing, prioritization of streams, header

[1]School of Information and Communication Technology, Hanoi University of Science and Technology
[2]Faculty of Computer Science, Phenikaa University
[3]Graduate School of Informatics, Chiba University, Japan
[*]Corresponding author, email: lenp@soict.hust.edu.vn
DOI: 10.56651/lqdtu.jst.v14.n01.1044.ict

compression, and server-initiated pushes. These features streamline protocol operations and enable effective resource allocation, prioritizing critical elements like Hyper Text Markup Language (HTML), Cascading Style Sheets (CSS), and scripts to improve rendering efficiency. This approach ensures that essential resources receive the necessary bandwidth to enhance user experience [3]–[5].

Further improvements are realized through QUIC [6], [7], a transport protocol developed to reduce latency, secure communications, and mitigate the limitations of running HTTP/2 over Transmission Control Protocol (TCP) and Transport Layer Security (TLS), specifically, handshake delays and head-of-line (HoL) blocking [8]. Even with these advances, the reliance on a single network interface remains a bottleneck for loading speeds. This challenge motivated the development of MPQUIC [9]–[11], multipath extensions for QUIC that leverage multiple network interfaces (e.g., Wi-Fi and LTE) simultaneously. By distributing traffic across multiple paths, MPQUIC not only increases aggregated bandwidth but also enables seamless transitions during network changes.

Although our work focuses on MPQUIC, the proposed methodology can be generalized to other multipath transport protocol implementations. Currently, popular browsers such as Chrome, Safari, and Firefox do not support MPQUIC, which restricts the direct deployment of HTTP/2 over MPQUIC [12] in real-world web browsing scenarios. Without native browser support, leveraging the full benefits of MPQUIC's multipath capabilities for enhancing web performance remains challenging. This limitation underscores the necessity for alternative approaches that can simulate and evaluate the potential advantages of MPQUIC without relying on immediate browser integration. To address this gap, this paper introduces a simulation platform that models web browsing with HTTP/2 over MPQUIC, enabling researchers to test scheduling strategies, optimize performance, and analyze multipath transmission in a controlled environment. Our main contributions are as follows:

- We introduce a novel framework for integrating HTTP/2 over MPQUIC, tackling key challenges in stream scheduling and protocol compatibility. Within this framework, we implement five stream schedulers: Round Robin (RR), Weighted Round Robin (WRR) [12], Scattered Weighted Round Robin (sWRR) [13], Stream-Aware Earliest Completion First (SA-ECF) [14], and a newly proposed data size-based WRR (dWRR).
- We thoroughly evaluate these schedulers using our framework in Mininet-WiFi. The results offer insights into the comparative performance of each scheduler and validate the effectiveness of the deployed framework.

The remainder of the paper is organized as follows. Section 2 reviews the related works, highlighting the challenges and existing solutions in HTTP/2 and MPQUIC integration. Section 3 details the design of our proposed framework, including its architecture and key components. Section 4 presents the results from our performance evaluation, followed by Section 5, which concludes the paper and suggests directions for future work.

2. Related works

2.1. Hypertext Transfer Protocol Version 2

Defined in RFC 7540 [1] and RFC 8740 [15], HTTP/2 represents a significant revision to the HTTP protocol to improve web performance through several key features. First, header compression reduces overhead by minimizing the size of frequently repeated headers, helping to accelerate page loads. Second, HTTP/2 permits true multiplexing, allowing multiple concurrent streams to share a single TCP connection. This approach reduces the time lost in setting up separate connections and mitigates HoL blocking within the application layer. Finally, HTTP/2 introduces stream prioritization mechanisms that let clients specify which resources are most critical. By coordinating bandwidth allocation, these features collectively enhance website responsiveness and optimize the usage of available network capacity.

2.2. HTTP/2 prioritization and dependency tree

A core feature of HTTP/2 is its prioritization framework [1], [4], which enables clients to specify the relative importance of different streams. This is achieved through two mechanisms: dependency trees and weight assignments.

Dependency trees: HTTP/2 organizes streams into a hierarchical structure where each stream can depend on another. For instance, a browser can indicate that CSS files depend on HTML files, meaning the HTML files should be fully loaded before the CSS files are fetched. Dependencies allow clients to define complex relationships between streams, ensuring that critical resources are loaded first.

Weight assignments: each stream is assigned a weight between 1 and 256, indicating its relative priority within its dependency group. Streams with higher weights receive a larger share of the available resources, ensuring they are processed more quickly.

These mechanisms allow HTTP/2 to allocate resources dynamically based on the application's needs. For example, a web page's primary HTML document can be given the highest priority, followed by CSS and JavaScript files, with images and videos receiving lower priority.

2.3. HTTP/2 over Multipath QUIC

The integration of HTTP/2 with MPQUIC has emerged as a promising solution to further improve web performance, particularly in heterogeneous networks where devices are equipped with multiple interfaces, such as Wi-Fi and LTE. MPQUIC extends the QUIC protocol by enabling simultaneous data transmission over multiple network paths, thereby increasing aggregate bandwidth, improving reliability, and facilitating seamless handovers during network transitions. Running HTTP/2 over MPQUIC offers the potential to leverage both application-layer and transport-layer enhancements. HTTP/2 provides stream multiplexing, prioritization, and header compression, while MPQUIC enables flexible use of multiple paths. When combined,

as shown in Figure 1, these two protocols can jointly support more efficient delivery of complex web pages, which often contain diverse resources (e.g., HTML, CSS, JavaScript, images, and videos) varying significantly in size and priority. However,

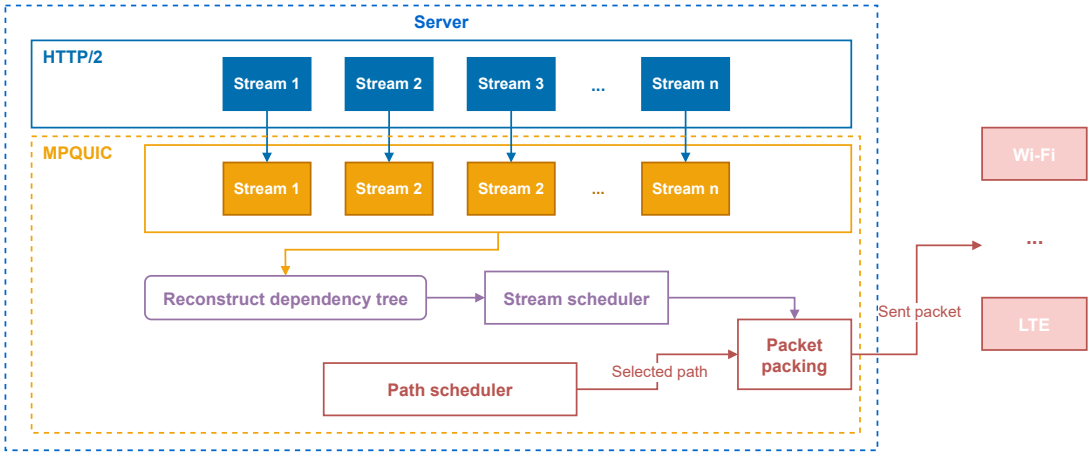


Fig. 1. HTTP/2 over Multipath QUIC.

this integration is not without challenges. One of the main obstacles is the lack of native support for MPQUIC in mainstream web browsers, including Chrome, Safari, and Firefox. As a result, deploying HTTP/2 over MPQUIC in real-world web browsing remains infeasible, limiting opportunities to evaluate and utilize its full potential in production environments. Another key challenge lies in resource scheduling, particularly coordinating the delivery of prioritized HTTP/2 streams across multiple QUIC paths. HTTP/2 is designed with a prioritization framework that specifies stream dependencies and relative weights, but this mechanism is originally intended for single-path TCP connections. When extended to MPQUIC, it becomes unclear how stream priorities should influence path selection and bandwidth allocation across concurrent paths. Moreover, the diverse characteristics of web objects, such as their size, importance, and loading sequence, introduce additional complexity for scheduling decisions.

2.4. Stream scheduling in MPQUIC

MPQUIC harnesses the combined capabilities of WLAN and cellular networks (e.g., LTE/4G/5G) on a single device to significantly enhance the robustness and performance of QUIC. By seamlessly integrating the multiplexing of application streams with the coordination of UDP sub-flows, MPQUIC optimizes the utilization of all available network paths, ensuring efficient and reliable data transmission [16]. Recent research in MPQUIC has explored multiple directions to enhance its deployment and efficiency. These include improvements in protocol architecture [10], [17], advancements in congestion control algorithms [11], [18], [19] to adapt to

varying path characteristics, path scheduling [20]–[22] strategies for selecting the best available network interface, and stream scheduling mechanisms to determine how data is transmitted across concurrent streams. Each component is crucial in delivering optimal performance, particularly in dynamic and heterogeneous networks. However, stream-level scheduling, especially in the context of HTTP/2's prioritization model, remains underexplored in terms of reusable and extensible implementations.

Over the years, several stream schedulers have been proposed to improve MPQUIC performance. Round Robin (RR) is a simple and fair scheduling method that allocates resources to streams cyclically. While RR ensures equal opportunities for streams, it fails to consider the size or priority of individual objects. As a result, RR can lead to inefficient resource utilization, especially when smaller objects or high-priority streams are delayed due to larger objects in the scheduling queue. Weighted Round Robin (WRR) [12] extends RR by incorporating stream priorities. In WRR, streams with higher priority values can send more packets per round. However, as shown in previous studies [8], this method is still susceptible to HoL blocking. For instance, streams with high priority values can consume the entire sending window, leaving smaller or high-priority streams unable to transmit. This issue is particularly problematic in HTTP/2, where object dependencies often require specific streams to be completed before others. Recent studies have explored stream-aware scheduling for MPQUIC to address these limitations. sWRR [13] is introduced to address the HoL blocking limitations of WRR. By scaling down the number of packets a stream can send each round, sWRR ensures that a few streams do not monopolize the sending window. This approach improves fairness and allows more streams to transmit in each scheduling cycle. Rabitsch *et al.* [14] introduce SA-ECF, a stream-aware scheduler for heterogeneous paths that considers path characteristics to reduce HoL blocking and improve fairness.

Despite these developments, existing works typically focus on individual scheduler designs without providing a unified platform to implement, evaluate, and compare them under realistic multipath network conditions. To the best of our knowledge, there is a lack of a general-purpose framework that integrates HTTP/2 over MPQUIC for stream scheduler research, especially one that supports real-world prioritization models such as browser-specific dependency trees and heterogeneous object types. In addition, existing stream schedulers (i.e, RR, WRR, sWRR, and SA-ECF) fail to effectively handle modern web traffic characterized by diverse object sizes and priorities. These methods often delay critical small objects, such as HTML and CSS, while larger, less essential streams dominate transmission. Additionally, WRR suffers from HoL blocking, where high-priority streams monopolize resources, preventing timely transmission of smaller, urgent streams. To overcome these issues, we propose dWRR, which groups streams by priority and sorts them by size within each group. By dynamically allocating scheduling opportunities based on weighted metrics, dWRR reduces latency for critical resources, mitigates HoL blocking, and adapts to changing network conditions, ensuring efficient and fair resource utilization in HTTP/2 over MPQUIC.

3. Framework design

3.1. Overview of the framework

Our framework integrates HTTP/2 and MPQUIC to address the challenges of stream-based scheduling and protocol compatibility in multipath networks. The framework is composed of the following key components:

Client-side: Simulates browser functionality with a dependency tree to manage HTTP/2 streams effectively. This module generates requests and tracks dependencies between streams.

Server-side: Reconstructs the dependency tree upon receiving client requests, ensuring streams are processed and prioritized based on their relationships and priorities.

Allocates HTTP/2 streams to multiple paths dynamically, considering real-time network conditions like RTT and bandwidth. It ensures that high-priority streams receive sufficient resources to minimize latency and improve performance.

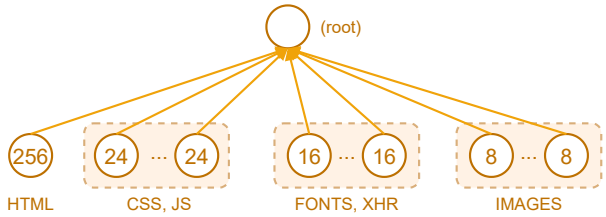
3.2. Reconstruct the dependency tree

Upon receiving requests from the client, the server reconstructs the dependency tree using metadata included in the client requests. Each request contains information about its dependency relationships, such as parent-child associations and priorities. The server interprets this data to rebuild the dependency hierarchy, as shown in Figure 2. The figure illustrates the stream dependency tree styles adopted by three major browsers: Safari, Firefox, and Chrome. Safari applies a flat structure where all resources (HTML, CSS/JS, fonts, XHR, images) are directly attached to the root, without strict ordering across types. Firefox constructs a deeper, dependency-aware tree where scripts and styles depend on the HTML node, and other resources (such as fonts and images) are children of those groups, reflecting the rendering logic. Chrome takes a grouped approach, organizing resources into prioritized batches, HTML and critical styles load first, followed by JavaScript and XHR, and finally images and pushed assets. This grouping also reflects Chrome's push strategy and staged loading behavior.

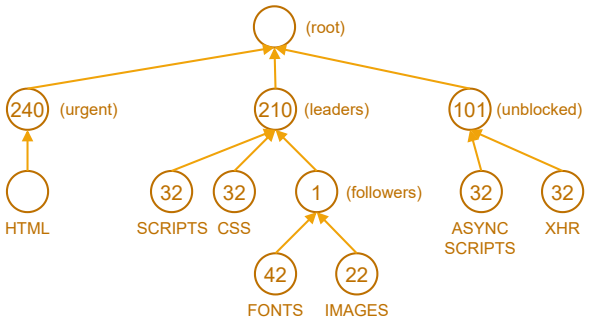
These browser-specific structures highlight the diversity in HTTP/2 prioritization logic, which must be preserved during reconstruction in order to reflect realistic loading behaviors. Our framework supports the emulation of each strategy by parsing the priority and dependency fields included in the requests and generating the corresponding dependency tree on the server side.

3.3. Stream schedulers

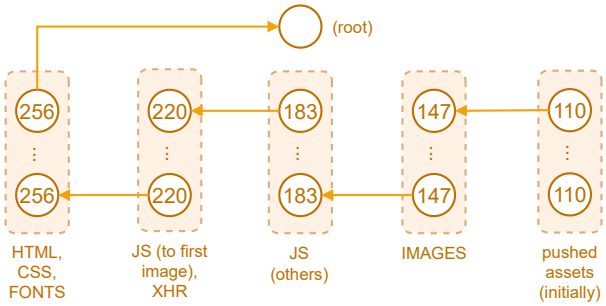
In addition to implementing previously proposed schedulers (including RR, WRR, SA-ECF, and sWRR), we implement dWRR. This new stream scheduler prioritizes streams based on both priorities and data size. The dWRR scheduler employs a structured process for selecting and transmitting stream packets to optimize performance in multipath networks.



(a) Safari



(b) Firefox



(c) Chrome

Fig. 2. HTTP/2 dependency tree style.

Initialization: the scheduler maintains a list of active streams, assigning each stream a priority and data size. Active streams with the same priority are grouped into one group called G_i.

Group sorting and prioritization: Figure 3 presents group sorting of dWRR. Within each priority group G_i, dWRR performs the following operations.

Sorting by data size: streams within G_i are sorted in ascending order of their data size. This strategy prioritizes the transmission of smaller streams, which can be completed quickly, thereby reducing dWRR and improving responsiveness.

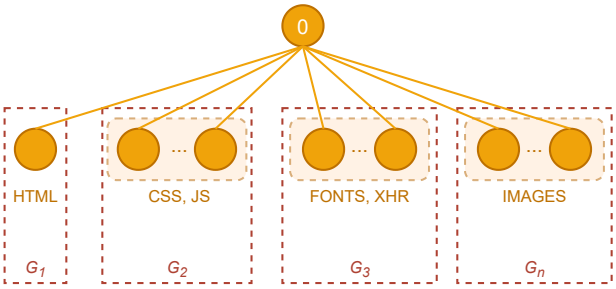


Fig. 3. The dWRR's group sorting.

Priority scaling: dWRR calculates a scaled priority value (p_i) for each group to determine its share of scheduling opportunities in the current round. The scaled priority is defined as:

$$p_i = min(1, \beta), \text{where } \beta = \frac{priority_{G_i}}{\sum_{j=1}^{N} priority_{G_j}}.$$

This formula ensures that a group's scheduling weight is proportional to its relative priority compared to all other groups while preventing any single group from monopolizing scheduling resources.

Stream selection in a round: after prioritization, dWRR selects streams for transmission in the current scheduling round by first considering the priority groups. Streams from higher-priority groups are processed before those in lower-priority groups, ensuring that critical streams are prioritized. Within each group, streams are further sorted and transmitted in ascending order of their data size, allowing smaller streams to be completed quickly and reducing delay.

3.4. Framework implementation

To simulate browser behavior in an HTTP/2 over MPQUIC environment, we build a client module based on the MPQUIC prototype source code [9]. We develop a client module responsible for sending concurrent requests to a set of URLs and tracking both the response times and the sizes of the received data. The module employs a WaitGroup mechanism to manage concurrent processing, ensuring that all requests are completed before proceeding to subsequent steps.

Before sending each request, the URL is assigned a priority parameter, including a weight and a stream dependency. The function classifies resources, mapping resource types such as HTML, CSS, images, fonts, or JavaScript to specific priority levels. Notably, the prioritization strategies are customized for each simulated browser (Safari, Firefox, Chrome) to reflect their real-world resource-handling behavior. For instance, Safari assigns the highest weight (255) to HTML resources to ensure rapid

loading of critical content, while Firefox incorporates stream dependency parameters to enforce a precise order of resource processing. The complete source code, with over 2000 lines of code added or modified, is available at https://github.com/quic-vn/HTTP-2-over-MPQUIC.

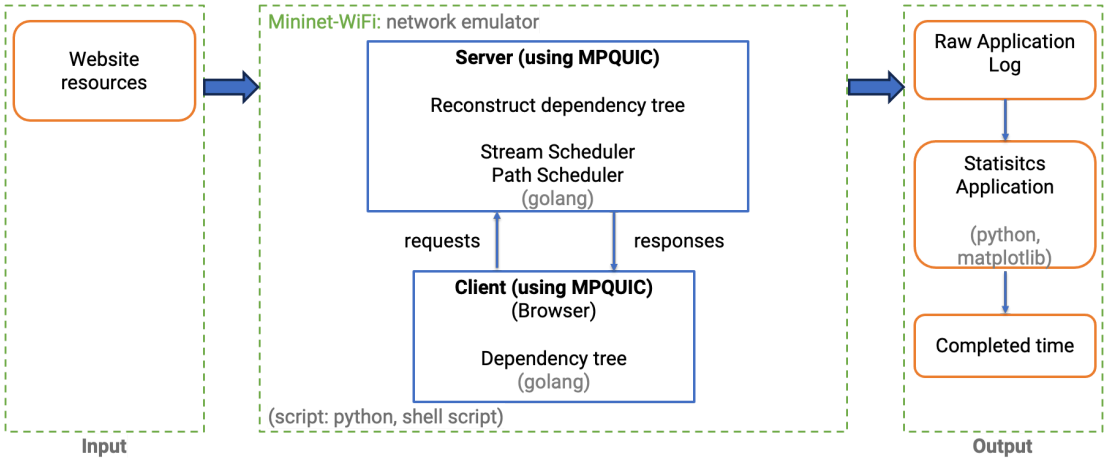


Fig. 4. Illustration of the HTTP/2 over MPQUIC framework implementation.

As depicted in Figure 4, the framework consists of three main components: *Input*, *Network emulator*, and *Output*.

- The *Input* module represents the website resources, which are fetched during the experiment.
- The *Mininet-WiFi network emulator* simulates real-world network conditions where the Server (using MPQUIC) handles stream scheduling and path scheduling while maintaining the dependency tree for HTTP/2 streams. The Client (using MPQUIC) acts as a browser, processing the dependencies and managing concurrent requests. Communication between the server and client is conducted through MPQUIC's multipath capabilities.
- The *Output* module logs raw application data, which is then processed by a Python-based statistics application to generate insights on download completion times and performance metrics.

4. Evaluation

4.1. Experiment settings

We use Mininet-WiFi [23] to emulate a multipath wireless network for our evaluations. Figure 5 illustrates the network model employed in our experiments. The emulated network topology comprises one server and a single mobile client within a 100 m × 100 m space, with a Wi-Fi access point at (45,50) and an LTE base station

at (55,50). The bandwidth for the wired connections is set within a range of 30 to 80 Mbps, referencing data from Ookla's Speedtest Global Index [24]. According to this source, the worldwide average download speed is 41.54 Mbps for mobile networks and 79 Mbps for broadband services. RTT configurations follow the guidelines provided by Google [25] and Microsoft [26]. Specifically, wired links experiencing medium levels of dynamicity are assigned an average RTT of 20 ms, while those with other dynamicity levels receive an average RTT of 30 ms. We employ TC and netem to accurately simulate wired links and dynamic network conditions. Netem is used with TC to configure random packet loss rates and RTT variations, as detailed in Table 1. For wireless connections, we use wmediumd alongside the Log Distance Propagation Loss Model to emulate realistic wireless networks. All experiments are conducted on a machine equipped with an Intel(R) Core(TM) i5-10400 CPU @ 2.90 GHz, 6 GB of RAM, running Ubuntu 22.04.

On the server, we deploy copies of the most visited web pages (according to [21]) that include files (i.e., HTML, CSS, JS, images, etc.) with different sizes, as shown in Table 2. We implement the proposed framework and compare stream schedulers, including RR, WRR, sWRR, SA-ECF, and dWRR.

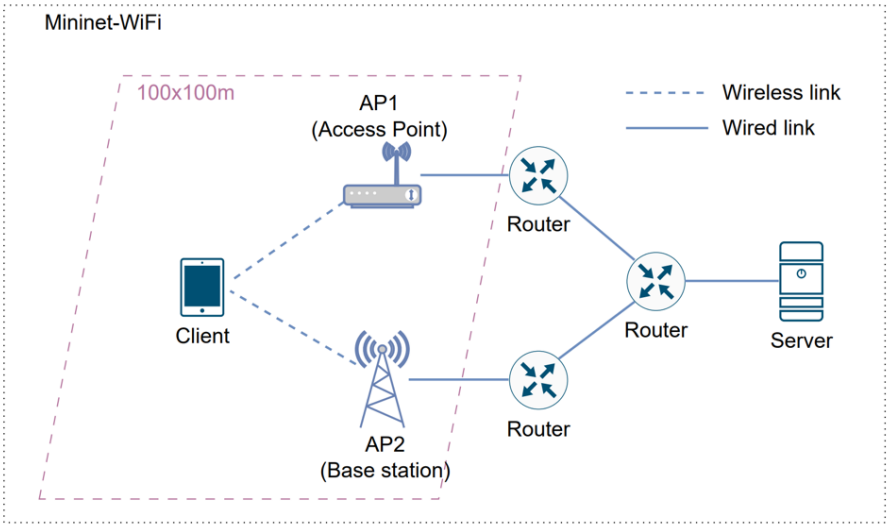


Fig. 5. Network model.

Table 1. Dynamicity levels

	Low	Medium	High
Delay variation (%)	0	8	16
Loss rate (%)	0	1.5	3

Table 2. Web page parameters

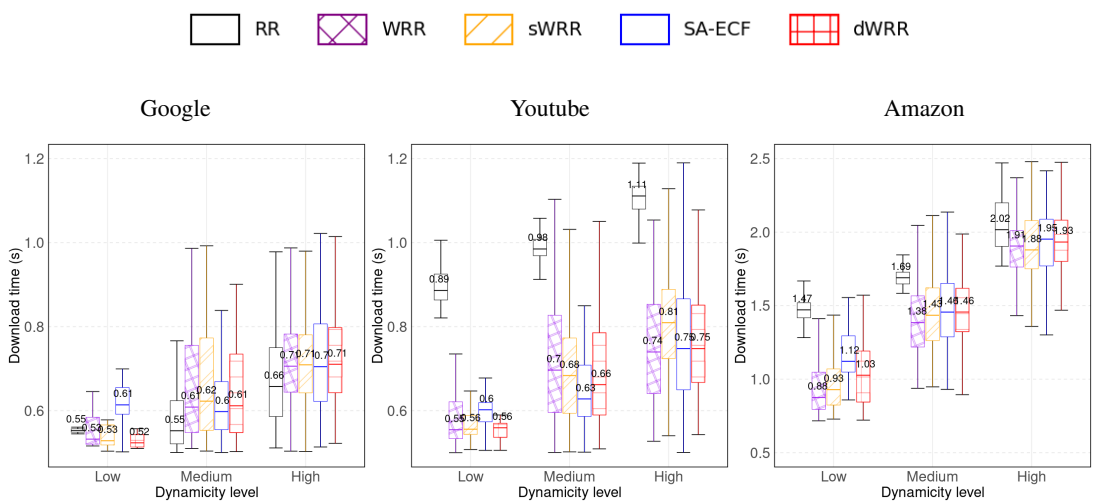
Website	Number of resources	Sizes
Google.com	15	1.7 MB
Youtube.com	51	1.6 MB
Amazon.com	25	5.0 MB

4.2. Results

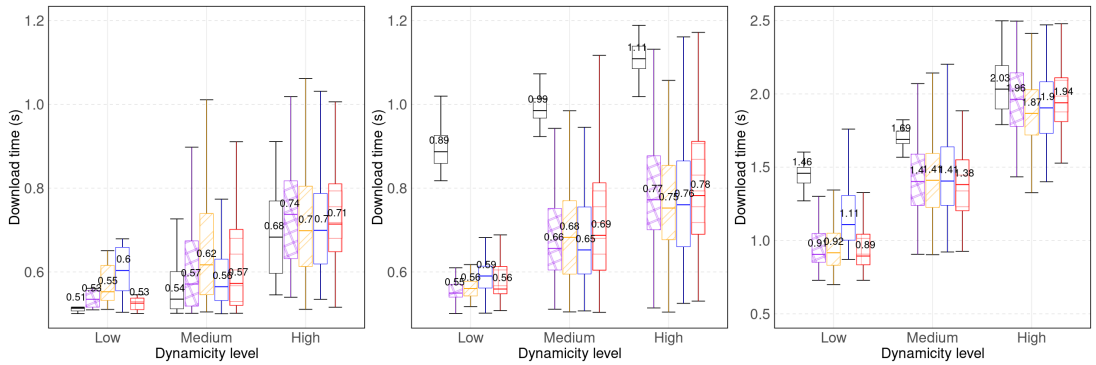
The framework validation: the results present in this section confirm the successful operation of the simulation framework for HTTP/2 over MPQUIC. Through comprehensive experiments conduct under varying network conditions, characterized by fluctuations in RTT and packet loss, and across different dependency tree structures (Safari, Firefox, and Chrome), the simulation framework effectively demonstrates its ability to replicate real-world multipath scheduling scenarios. The observed metrics, including web page download time and object completion time, validate the framework's reliability and capability in analyzing the performance of schedulers under diverse conditions.

Web page loading: we record the download time until the client receives all the files. Figure 6 illustrates the download times for different schedulers when accessing web pages under varying levels of dynamicity, defined by the variation in RTT and packet loss rate. As the dynamicity level increases, download times consistently rise for all schedulers. This demonstrates that higher variability in network conditions adversely impacts the efficiency of data transmission and increases the overall time required to load the webpage, showing that these schedulers are not yet able to adapt in dynamic networks. RR scheduler achieves the shortest download times when dealing with a small number of objects, such as in the case of Google.com. However, as the number of objects increases, RR's performance degrades more significantly compared to other schedulers. For Youtube.com, RR's download time increases rapidly to 1.1 seconds, which is considerably higher than the range of 0.56–0.7 seconds observed with other schedulers. Similarly, for Amazon.com, where the number of objects is significantly larger, RR's download time rises to 2.03 seconds, while other schedulers manage to stay within a range of 0.91–1.94 seconds. These results highlight the limitations of RR in handling scenarios with a large number of objects under variable network conditions. Its simplistic scheduling mechanism does not efficiently adapt to the increased complexity introduced by high RTT and packet loss rates, leading to longer download times compared to its counterparts.

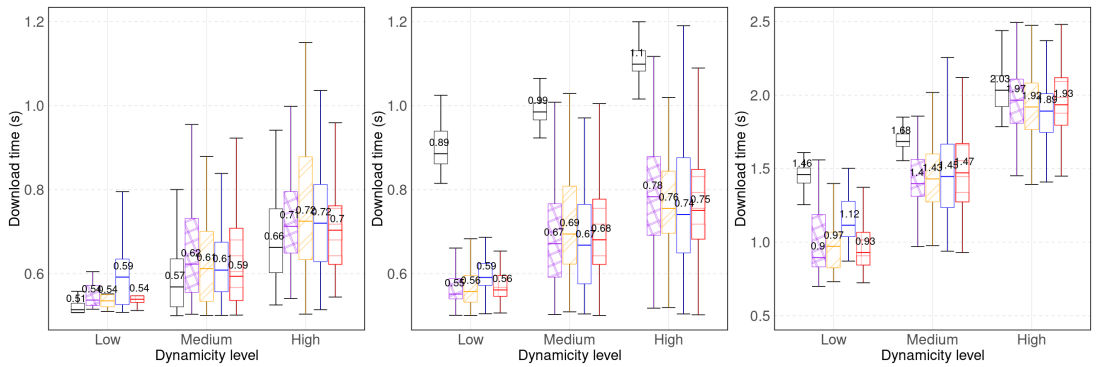
In the different dependency tree structures, the trends suggest no significant differences in web page loading times across the different dependency tree structures. However, specific cases highlight how particular stream schedulers may perform better or worse compared to others, depending on the conditions. For example, when browsing Amazon.com, the dWRR scheduler achieves the best performance under Firefox's dependency tree structure (low and medium dynamicity level). In contrast,



(a) Safari's dependency tree style



(b) Firefox's dependency tree style

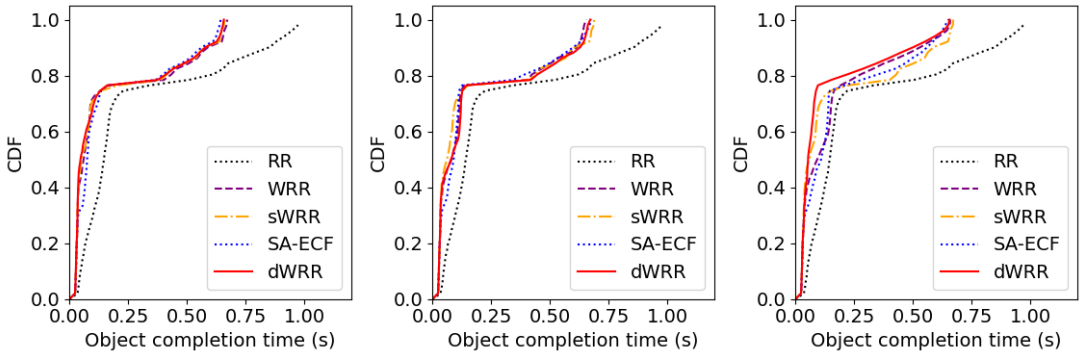


(c) Chrome's dependency tree style

Fig. 6. Web page loading comparison for different browsers and major websites.

its performance is 3–12% lower under Safari's dependency tree structure compared to other schedulers. This indicates that dWRR's efficiency is influenced by the interaction between the dependency tree style and the number of objects, along with the variability in RTT and packet loss.

Object completion time: we also perform calculations of object completion time [27] to evaluate the web page loading improvement process. The object completion time represents a cumulative time-based metric aggregating data from object downloading completion events, offering a detailed perspective on how web page load times influence the user experience. Figure 7 depicts the Cumulative distribution function (CDF) of object completion times across all objects when accessing Youtube.com under a medium level of dynamicity, evaluate with three dependency tree styles: Safari, Firefox, and Chrome. The object completion time provides a cumulative perspective, aggregating data from object download events to highlight how different schedulers affect user-perceived loading times. RR scheduler consistently displays a slower performance, as indicated by its flatter CDF curve across all dependency tree styles. This suggests that RR takes longer to complete object downloads, particularly for later objects, resulting in a less efficient user experience. In contrast, WRR, sWRR, SA-ECF, and dWRR exhibit steeper CDF curves, with significant portions of objects completing faster. Among these, dWRR achieves the best performance in the dependency tree style of Chrome (the world's most popular browser), where it outperforms other schedulers by ensuring a higher percentage of objects are completed in a shorter period of time.



(a) Safari's dependency tree style *(b) Firefox's dependency tree style* *(c) Chrome's dependency tree style*

Fig. 7. Object completion time.

5. Conclusion and future works

In the paper, we present a framework for integrating HTTP/2 over MPQUIC, aiming to harness multipath capabilities to enhance web browsing performance. The approach addresses challenges in stream scheduling by proposing four distinct scheduling strategies (RR, WRR, sWRR, SA-ECF, and dWRR) and implementing

them within a controlled simulation environment based on Mininet-WiFi. Through extensive experiments, we demonstrate the framework's effectiveness in capturing realistic network conditions and assessing scheduler performance under varying dynamicity levels. Our results highlight that optimized scheduling can substantially improve key metrics such as page load time and resource allocation. Moreover, the experiments confirm the feasibility and reliability of deploying an HTTP/2 over MPQUIC architecture in simulated multipath networks.

Despite these advancements, this work opens up several avenues for future exploration. First, further validation of the proposed framework in more diverse network topologies, such as scenarios with multiple clients and multiple servers, would provide deeper insights into real-world scalability. Second, extending the scheduling algorithms to account for more nuanced factors (e.g., energy consumption on mobile devices or application-level quality-of-service requirements) could improve the adaptability of the framework to heterogeneous networks. Finally, porting our approach to real browser platforms, once MPQUIC support becomes mainstream, would enable direct comparisons against native QUIC and other transport-layer implementations, accelerating the adoption of multipath solutions in large-scale web ecosystems.

Acknowledgement

Thanh Trung Nguyen was funded by the Master, PhD Scholarship Programme of Vingroup Innovation Foundation (VINIF), code VINIF.2024.TS.065.

References

[1] M. Belshe, R. Peon, and M. Thomson, "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, 2015. DOI: 10.17487/RFC7540

[2] B. Pollard, *HTTP/2 in action*. Shelter Island, NY: Manning, 2019.

[3] J. Nakazato, K. Nakagawa, K. Itoh, R. Fontugne, M. Tsukada, and H. Esaki, "WebRTC over 5G: A study of remote collaboration QoS in mobile environment," *Journal of Network and Systems Management*, vol. 32, 2024. DOI: 10.1007/s10922-023-09778-5

[4] M. Wijnants, R. Marx, P. Quax, and W. Lamotte, "HTTP/2 prioritization and its impact on web performance," in *Proceedings of the 2018 World Wide Web Conference*, Republic and Canton of Geneva, Switzerland, 2018, pp. 1755–1764. DOI: 10.1145/3178876.3186181

[5] R. Kohavi, A. Deng, R. Longbotham, and Y. Xu, "Seven rules of thumb for web site experimenters," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, 2014, pp. 1857–1866. DOI: 10.1145/2623330.2623341

[6] J. Iyengar and M. Thomson, "QUIC: a UDP-based multiplexed and secure transport," RFC 9000, 2021. DOI: 10.17487/RFC9000

[7] M. Thomson and S. Turner, "Using TLS to secure QUIC," RFC 9001, 2021. DOI: 10.17487/RFC9001

[8] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang, and Z. Shi, "The QUIC transport protocol: design and Internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, New York, NY, USA, 2017, pp. 183–196. DOI: 10.1145/3098822.3098842

[9] Q. De Coninck and O. Bonaventure, "Multipath QUIC: design and evaluation," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, New York, NY, USA, 2017, pp. 160–166. DOI: 10.1145/3143361.3143370

[10] T. Viernickel, A. Froemmgen, A. Rizk, B. Koldehofe, and R. Steinmetz, "Multipath QUIC: a deployable multipath transport protocol," in *2018 IEEE International Conference on Communications*, 2018, pp. 1–7. DOI: 10.1109/ICC.2018.8422951

[11] B. Kimura, S. Ferlin, T. Paiva, T. Mahmoodi, A. Brunstrom, and O. Alay, "Evaluating adaptive video streaming over Multipath QUIC with shared bottleneck detection," *ACM Transactions on Multimedia Computing, Communications, and Applications*, 2025. DOI: 10.1145/3711862

[12] J. Wang, Y. Gao, and C. Xu, "A Multipath QUIC scheduler for mobile HTTP/2," in *Proceedings of the 3rd Asia-Pacific Workshop on Networking*, New York, NY, USA, 2019, pp. 43–49. DOI: 10.1145/3343180.3343185

[13] Y. Xing, K. Xue, Y. Zhang, J. Han, J. Li, D. S. L. Wei, R. Li, Q. Sun, and J. Lu, "A stream-aware MPQUIC scheduler for HTTP traffic in mobile networks," *IEEE Transactions on Wireless Communications*, vol. 22, no. 4, pp. 2775–2788, 2023. DOI: 10.1109/TWC.2022.3213638

[14] A. Rabitsch, P. Hurtig, and A. Brunstrom, "A stream-aware Multipath QUIC scheduler for heterogeneous paths," in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, New York, NY, USA, 2018, pp. 29–35. DOI: 10.1145/3284850.3284855

[15] D. Benjamin, "Using TLS 1.3 with HTTP/2," RFC 8740, 2020. DOI: 10.17487/RFC8740

[16] Y. Liu, Y. Ma, Q. D. Coninck, O. Bonaventure, C. Huitema, and M. Kühlewind, "Multipath extension for QUIC," Internet Engineering Task Force, Tech. Rep., 2025.

[17] M. Franke, J. Holland, and S. Schmid, "MCQUIC – a multicast extension for QUIC," *arXiv*, 2023. DOI: 10.48550/arXiv.2306.17669

[18] W. Yang, L. Cai, S. Shu, and J. Pan, "Mobility-aware congestion control for Multipath QUIC in integrated terrestrial satellite networks," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 11 620–11 634, 2024. DOI: 10.1109/TMC.2024.3397164

[19] Z. Deng, Y. Liu, J. Liu, A. Argyriou, and D. Liu, "BBR-based and fairness-guaranteed congestion control and packet scheduling for MPQUIC over heterogeneous networks," *Computer Communications*, vol. 224, pp. 213–224, 2024. DOI: 10.1016/j.comcom.2024.06.006

[20] P. S. Kumar, P. Saxena, and Ö. Alay, "DEAR: DRL empowered actor-critic scheduler for Multipath QUIC under 5G/B5G hybrid networks," in *Advanced Information Networking and Applications - Proceedings of the 38th International Conference on Advanced Information Networking and Applications*, Kitakyushu, Japan, 2024, pp. 103–113. DOI: 10.1007/978-3-031-57840-3_10

[21] T. T. Nguyen, M. H. Vu, T. H. L. Dinh, T. H. Nguyen, P. L. Nguyen, and K. Nguyen, "FQ-SAT: A fuzzy Q-learning-based MPQUIC scheduler for data transmission optimization," *Computer Communications*, vol. 226-227, 2024. DOI: 10.1016/j.comcom.2024.107924

[22] X. Zhang, Y. Zhang, P. Dong, D. Yang, X. Du, C. Yu, and H. Zhang, "TA2LS: a traffic-aware multipath scheduler for cost-effective QoE in dynamic HetNets," *IEEE Transactions on Mobile Computing*, vol. 23, no. 12, pp. 13 603–13 620, 2024. DOI: 10.1109/TMC.2024.3440415

[23] R. d. R. Fontes and C. E. Rothenberg, "Mininet-WiFi: a platform for hybrid physical-virtual software-defined wireless networking research," in *Proceedings of the 2016 ACM SIGCOMM Conference*, New York, NY, USA, 2016, pp. 607–608. DOI: 10.1145/2934872.2959070

[24] Ookla, "The Speedtest Global Index," [Online]. Available: https://speedtest.net/global-index, 2024, [Accessed: 2024-10-30].

[25] Google, "Google Meet network requirements," [Online]. Available: https://support.google.com/a/answer/1279090, 2024, [Accessed: 2024-10-30].

[26] Microsoft, "The media quality and network connectivity performance," [Online]. Available: https://learn.microsoft.com/en-us/microsoftteams/prepare-network, 2024, [Accessed: 2024-10-25].

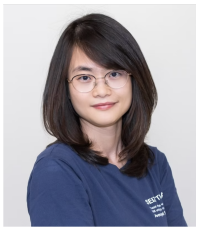
[27] E. Bocchi, L. De Cicco, and D. Rossi, "Measuring the quality of experience of Web users," in *Proceedings of the 2016 Workshop on QoE-Based Analysis and Management of Data Communication Networks*, New York, NY, USA, 2016, pp. 37–42. DOI: 10.1145/2940136.2940138

Manuscript received 31-12-2024; Accepted 15-06-2025.

■

Thanh Trung Nguyen received his B.E. and M.S. degrees from Hanoi University of Science and Technology, Vietnam, in 2016 and 2019, respectively. Currently, he is a Ph.D. student at the School of Information and Communication Technology at Hanoi University of Science and Technology. His research interests include network architecture, multipath transport protocol, optimization, and reinforcement learning. Email: trung.nt220001D@sis.hust.edu.vn

Minh Hai Vu received his Bachelor of Engineering degree from the School of Information and Communication Technology, Hanoi University of Science and Technology, in 2024. His research interests include reinforcement learning, the application of artificial intelligence in computer networking, and wireless communication systems. His work on assessing and optimizing multipath schedulers for mobile networks has been published in flagship conferences (e.g., IEEE CCNC, IEEE VTC-Fall). Email: hai.vm194550@sis.hust.edu.vn



Phi Le Nguyen received her B.E. and M.S. degrees from the University of Tokyo in 2007 and 2010, respectively. She received her Ph.D. in Informatics from The Graduate University for Advanced Studies, National Institute of Informatics, Tokyo, Japan in 2019. Currently, she is an Associate Professor at the School of Information and Communication, Hanoi University of Science and Technology, Vietnam. Her research interests include network architectures, applied AI in various domains such as smart healthcare, environement, and next generation networks. Email: lenp@soict.hust.edu.vn

Kien Nguyen (SM'16) received a B.E. degree in electronics and telecommunication from the Hanoi University of Science and Technology (HUST), Vietnam, in 2004, and a Ph.D. degree in informatics from the Graduate University for Advanced Studies, Japan in 2012. Dr. Nguyen was a researcher at the National Institute of Information and Communication Technology (NICT), Japan during 2014-2018. Since 2018 he has been with Chiba University, where he is currently an associate professor. His research covers a wide range of topics in networking and distributed systems, including the Internet, the Internet of Things technologies, and distributed ledger technologies. His research achievements have been disseminated in three patents, several IETF Internet drafts, and more than 160 publications in peer-reviewed journals and conferences. He is a senior member of IEEE and a member of IEICE, IPSJ. He also participates in IETF activities. Email: nguyen@chiba-u.jp

MỘT NỀN TẢNG MỚI CHO HTTP/2 TRÊN MPQUIC: THIẾT KẾ VÀ TRIỂN KHAI

Nguyễn Thành Trung, Vũ Minh Hải, Nguyễn Phi Lê, Nguyễn Kiên

Tóm tắt

Nhu cầu ngày càng tăng đối với truy cập web hiệu suất cao và đáng tin cậy đã thúc đẩy những tiến bộ đáng kể trong các giao thức truyền tải. Nghiên cứu này giới thiệu một khung nền tảng để triển khai HTTP/2 qua giao thức đa đường Multipath QUIC (MPQUIC), được thiết kế để nâng cao hiệu quả truyền dữ liệu trong các mạng không dây không đồng nhất. Bằng cách tích hợp khả năng ghép kênh và ưu tiên của HTTP/2 với khả năng sử dụng nhiều đường dẫn mạng đồng thời của MPQUIC, nền tảng này giải quyết các thách thức quan trọng trong việc lập lịch luồng và khả năng tương thích của giao thức. Để xác thực tính thực tiễn của nó, nhóm tác giả thiết kế và triển khai một số trình lập lịch luồng trong nền tảng. Các đánh giá thử nghiệm toàn diện trong trình giả lập Mininet-WiFi xác nhận hiệu quả của nền tảng và cung cấp những hiểu biết so sánh về hiệu suất của trình lập lịch. Nghiên cứu này đặt nền tảng vững chắc cho sự phát triển trong tương lai của trình duyệt web hiệu suất cao qua MPQUIC.

Từ khóa

HTTP/2; giao thức MPQUIC; nền tảng; lập lịch luồng.