Enviado: 11/06/2024 *Recibido: 11/06/2024* *Aceptado: 18/07/2024*

UNIVERSIDAD DISTRITAL FRANCISCO JOSE DE CALDAS

VISIÓN ELECTRÓNICA

Algo más que un estado sólido

https://doi.org/10.14483/issn.2248-4728

VISIÓN ELECTRÓNICA

Evaluation of Throughput in Live Video Streaming: TCP vs. QUIC as a Proof of Concept

Evaluación de throughput en transmisión de video en vivo: TCP vs. QUIC como prueba de concepto

Edith Paola Estupiñan Cuesta[1], Juan Carlos Martínez[2], William Andrés Roldán Pineda[3]

Abstract

The QUIC (Quick UDP Internet Connections) protocol has evolved to be standardized by the IETF (Internet Engineering Task Force), standing out for addressing inherent issues in TCP (Transmission Control Protocol). This article focuses on the use of QUIC in video transmission through MoQ (Media over QUIC), comparing its performance with the combination of RTMP (Real Time Messaging Protocol) and HLS (HTTP Live Streaming) over TCP, and evaluating the throughput of each protocol. A cloud-based test scenario was designed with two relay servers: one operating with MoQ and another with RTMP+HLS over TCP. Both were implemented as proof-of-concept developments in GO. Two clients, one local in the cloud and

[1] Telecommunications Engineer, graduate of Universidad Militar Nueva Granada, Colombia. Master's in Electronic Engineering from Pontificia Universidad Javeriana, Colombia. Professor at Universidad Militar Nueva Granada, Bogotá, Colombia, in the Telecommunications Engineering Program. E-mail: edith.estupinan@unimilitar.edu.co ORCID: https://orcid.org/0000-0002-4100-4943

[2] Electronic Engineer, graduate of Universidad Manuela Beltrán, Colombia. Specialist in Physical and Computer Security, Escuela de Comunicaciones Militares, Colombia. Master's in Automatic Production Systems from Universidad Tecnológica de Pereira, Colombia. Professor at Universidad Militar Nueva Granada, in the Telecommunications Engineering Program. E-mail: juan.martinezq@unimilitar.du.co ORCID: https://orcid.org/0000-0001-9893-6592

[3] Telecommunications Engineer Student. Universidad Militar Nueva Granada, Bogotá, Colombia. E-mail: est.william.roldan@unimilitar.edu.co

another remote, performed the publishing and subscription of a video with a bit rate of 8 Mbps and 720p quality, aiming to identify the performance differences between both scenarios. The tests were conducted at different times of the day and for various durations, alternating between local and remote scenarios, while traffic was captured using Wireshark and tcpdump. The results showed a channel utilization of 8.16 Mbps for RTMP + HLS over TCP and 0.6 Mbps for MoQ in publishing, and 1.04 Mbps and 0.3 Mbps respectively in subscription. Making MoQ superior to RTMP + HLS for its low throughput utilization thanks to the use of relay servers and the publisher/subscriber model.

Keywords: HLS, MoQ, Performance, QUIC, RTMP, TCP

Resumen

El protocolo QUIC (Quick UDP Internet Connections) ha evolucionado hasta ser estandarizado por la IETF (Internet Engineering Task Force), destacándose por resolver problemas inherentes a TCP (Transmission Control Protocol). Este artículo se enfoca en el uso de QUIC en la transmisión de video mediante MoQ (Media over QUIC), comparando su rendimiento con la combinación de RTMP (Real Time Messaging Protocol) y HLS (HTTP Live Streaming) sobre TCP, y evaluando el throughput de cada protocolo. Se diseñó un escenario de prueba en la nube con dos servidores de retransmisión: uno operando con MoQ y otro con RTMP + HLS sobre TCP. Ambos fueron implementados como pruebas de concepto desarrolladas en GO. Dos clientes, uno local en la nube y otro remoto, realizaron la publicación y suscripción de un video con un bitrate de 8 Mbps y calidad 720p, con el fin de identificar las diferencias de rendimiento entre ambos escenarios. Las pruebas se llevaron a cabo en diferentes momentos del día y con distintas duraciones, alternando entre escenarios locales y remotos, mientras se capturaba el tráfico con Wireshark y tcpdump. Los resultados mostraron una utilización de canal de 8.16 Mbps para RTMP+HLS sobre TCP y 0.6 Mbps para MoQ en la publicación, y de

1.04 Mbps y 0.3 Mbps respectivamente en la suscripción. Haciendo que MoQ sea superior a RTMP + HLS debido a su poca utilización de canal gracias al uso de servidores retransmisores y su modelo publicador/suscriptor.

Palabras clave: HLS, MoQ, Rendimiento, QUIC, RTMP, TCP

1. Introduction

In the transport layer of the OSI model, UDP and TCP have dominated for over 40 years since their respective publications in 1980 and 1981. Their widespread adoption has shaped the internet as we know it today, marked by remarkable evolution due to the exponential growth of internet traffic and the need for fast and efficient data transfer [1] [2]. TCP, a protocol providing a reliable and ordered connection for applications, has seen performance improvements through reduced retransmission and connection establishment times thanks to TCP's optional parameter field that has been used to enhance network behavior and support specific features that boost overall performance [3] [4] [5].

Advances in transport protocols have enabled HTTP to evolve similarly, allowing for persistent connections and multiple requests over a short period. This progression reduced network congestion and latency, making connection optimization clear [6]. Following the first HTTP version, further optimizations in resource usage and latency perception emerged, allowing for multiple exchanges within the same connection and encryption enhancements through TLS1.3 [7]. However, TCP's ordered packet transmission faces "head-of-line blocking," where a single packet retransmission stalls the entire connection [8].

Explorations of faster data transfer options began before HTTPv2, with researchers examining unreliable protocols. In 2013, Google aimed to speed up web page load times, leading to the creation of QUIC, a reliable protocol operating over UDP that minimizes latency through optimized network resource usage. Google began standardizing QUIC in 2017, presenting the

protocol at SIGCOMM'17 [9]. QUIC was designed to address TCP's head-of-line blocking and three-way handshake latency, expediting client-server data exchange [10].

Since its announcement, QUIC has been adopted as a transport protocol for a wide range of content distribution services, with numerous browsers and applications supporting it, and many others beginning to adopt it. QUIC's performance has been evaluated over the past few years to determine how much faster it is compared to TCP. Studies such as [11] include performance tests conducted while the protocol was still undergoing improvements, and results indicated that QUIC had better channel utilization optimization than TCP. However, at that stage, significant advantages were not yet evident when network variables like jitter were considered. The QUIC protocol reached its first official version in 2021, under RFC 9000 [12], and now serves as the core protocol for HTTP in its third version. This progress indicates strong acceptance of QUIC's use and its potential for widespread adoption [13].

Similarly, in previous studies, Eva Gagliardi et al. [14] explored QUIC and its various characteristics, from the composition of its connections and the encapsulation of TLS in CRYPTO frames to its packet structure. This study highlights testing conducted on different public servers with various implementations of QUIC draft-23, where they proposed the first framework designed to observe this protocol's behavior during connection establishment. On the other hand, Konrad Wolsing et al. [15] conducted a comparison between the TCP + TLS + HTTP/2 stack and QUIC, aiming to identify QUIC's strengths relative to the full suite of TCP web optimizations. They evaluated web performance metrics such as page load times and visual changes on the pages. Results showed that QUIC could load page content up to 131.3 ms faster than TCP on DSL networks and 345 ms faster on LTE, thanks to its optimized connection establishment, use of CUBIC for congestion control, and elimination of head-of-line blocking enhancing performance on unstable or lossy networks. Study [16] evaluated the use of QUIC for real-time video streaming through a partially adapted DASH framework integrated

with QUIC using HTTP/3. The tests recorded throughput data over two 5G links during a video streaming session where a person alternately walked and ran, showing channel utilization between 1.2 Mbps while walking and 1.7 Mbps while running.

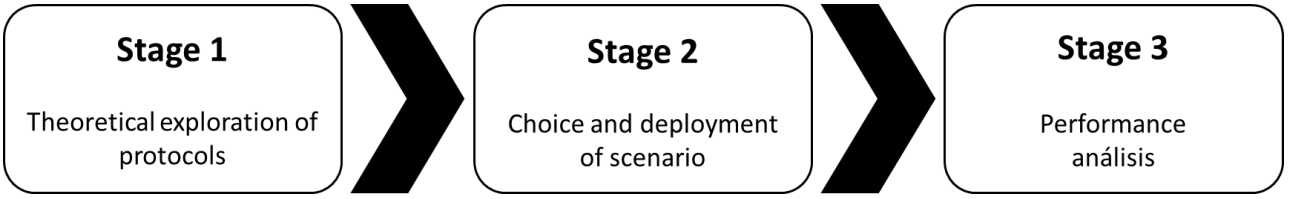
Considering prior research on QUIC vs. TCP performance evaluations in terms of content load speeds on web pages and throughput utilization in frameworks like DASH, this document presents a performance analysis of QUIC and TCP for HTTP use in real-time video streaming. Proof-of-concept tests were developed in GO with various protocols, focusing on throughput measurements to compare the strengths of each. This document evaluates throughput for the MoQ and RTMP + HLS protocols in real-time video streaming through proof-of-concept implementations in GO. A distinguishing feature of this study is the deployment of services in a public cloud environment to identify differences in channel utilization in both remote and local cloud-based settings. During testing, throughput for each protocol was measured throughout the publishing and consumption of the transmission.

Finally, this article begins with a contextual introduction; section 2 presents the proposed methodology for conducting the research. Section 3 details the obtained results and their subsequent analysis, and section 4 presents the conclusions of the study.

2. Methodology

The research is based on a descriptive experimental methodology divided into three stages, as shown in Figure 1.

Figure 1. Methodology stages



Source: Own work

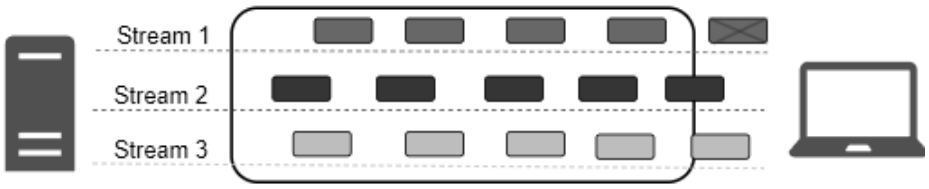
2.1 Theoretical Exploration of Protocols

In the first stage, a theoretical exploration of the QUIC and TCP protocols was conducted, alongside the protocols intended for the proof-of-concept tests, namely MoQ and RTMP + HLS.

2.1.1 QUIC

It is a reliable transport layer protocol that operates over UDP, providing structured communication based on controlled stream flows. Among its advantages is the avoidance of head-of-line blocking present in TCP, achieved through the multiplexing of multiple streams within a single connection without the need to deliver them in an organized manner. This allows for fewer retransmissions in the connection and substantially reduces latency, as each stream is independent of the others and has a unique identifier that distinguishes it from the rest, as shown in Figure 2 [17].

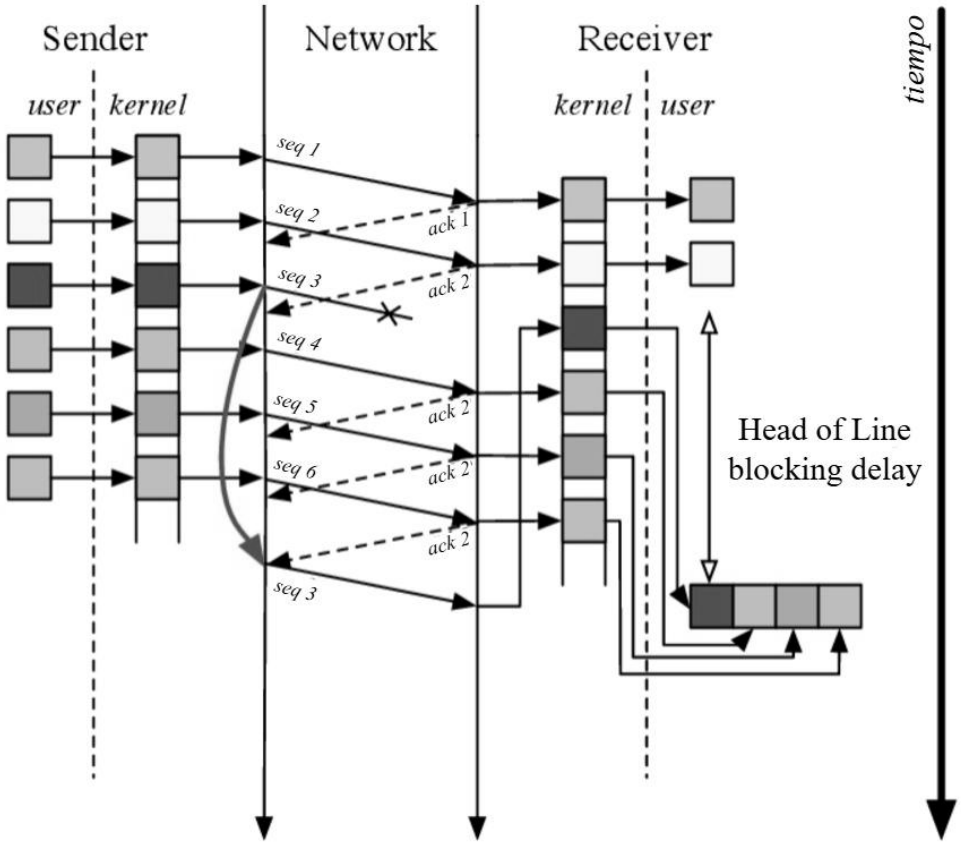
Figure 2. Stream multiplexing in QUIC



Source: Own work

As evidenced in Figure 3, in TCP, head-of-line blocking causes the connection to be slowed down because it is a reliable protocol that delivers packets in order. Therefore, until the lost packet is retransmitted, the queued packets cannot be transmitted.

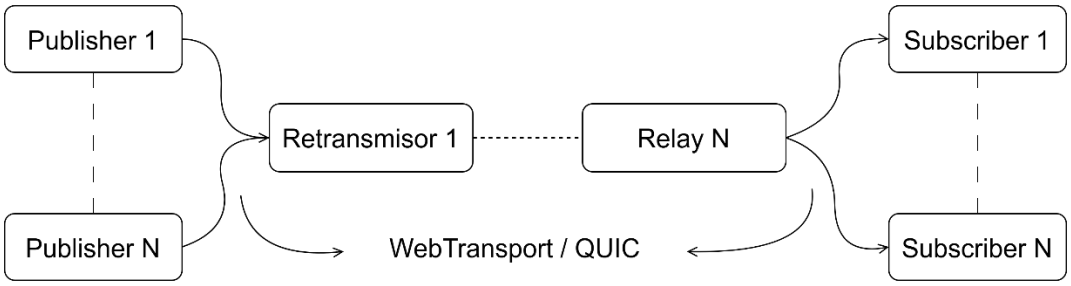
Figura 3. Bloqueo de cabeza de línea TCP [18]



2.1.2 Media over Quic (MoQ)

It is a protocol specifically optimized for QUIC, either directly or using WebTransport in browsers [19]. This protocol takes advantage of all the benefits of QUIC for low-latency media streaming through the use of a publisher/subscriber model, where the publisher broadcasts in response to multiple requests from different endpoints via servers acting as retransmitters, which forward the stream to other retransmitters or subscribers, as shown in Figure 4. There is no need to add unique encoding for each receiver; each stream is assigned a unique identifier that distinguishes it from the others, which the subscriber uses to consume the stream [20].

Figure 4. End-to-end deployment diagram of MoQ [21].



WebTransport is a framework that allows clients constrained by the web security model to communicate with a remote server through a secure multiplexed transport, utilizing the stream multiplexing feature of QUIC [22].

2.1.3 Throughput

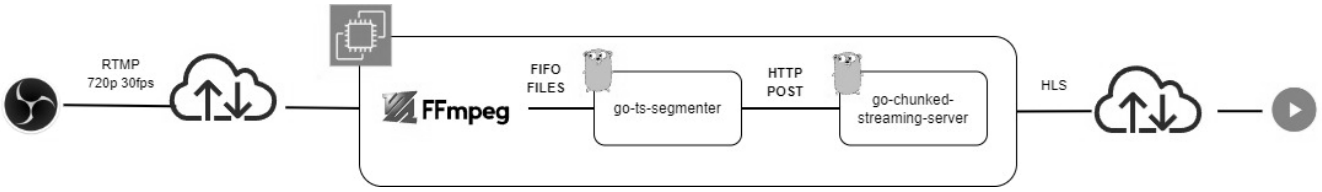
It is the actual speed at which data is successfully delivered over a network connection and provides a realistic measure of network performance. This metric is typically measured in bits per second (bps).

2.2 Choice and deployment of scenario.

The implementation proposed in this document for MoQ is based on a proof of concept developed in the GO programming language. In this setup, the browser itself is used as both the publisher and the subscriber, and the entire stream transmission is sent via QUIC through the WebTransport framework [23]. The protocols used for comparison are MoQ and HLS. HLS is a reliable protocol used for continuous video delivery, allowing the receiver to adapt the bit rate to network conditions in order to maintain uninterrupted playback with the best possible quality. Its use has been globally adopted since the protocol was announced in 2009 [24].

This protocol is commonly used in video streaming services that require low latency without compromising quality. RTMP (Real-Time Messaging Protocol) is used to process decoded video data, which, in this case, is generated by OBS, then segmented and published to a web server using live fragmented transfer of multimedia frames. Finally, the subscriber makes the stream requests, and if the requested object has not been fully received, the server will send all available parts of the object and keep the GET request open, sending the remaining fragments in real time until the object is fully transmitted and the request is closed. Figure 5 illustrates the behavior of RTMP + HLS as described above and the reference for the scenario to be implemented [25].

Figure 5. RTMP + HLS probe of concept [15].



The proposed scenario consists of a cloud environment made up of two servers running Ubuntu Server 20.04, on which two proof-of-concept implementations developed in GO are deployed, taken from [13] and [15], with the aim of delivering low-latency video streaming. Both streaming services operate over a web service that supports responding to subscriber requests, which can be validated in the tests performed, summarized in Table 1.

Table 1. Proposed test configuration and environments.

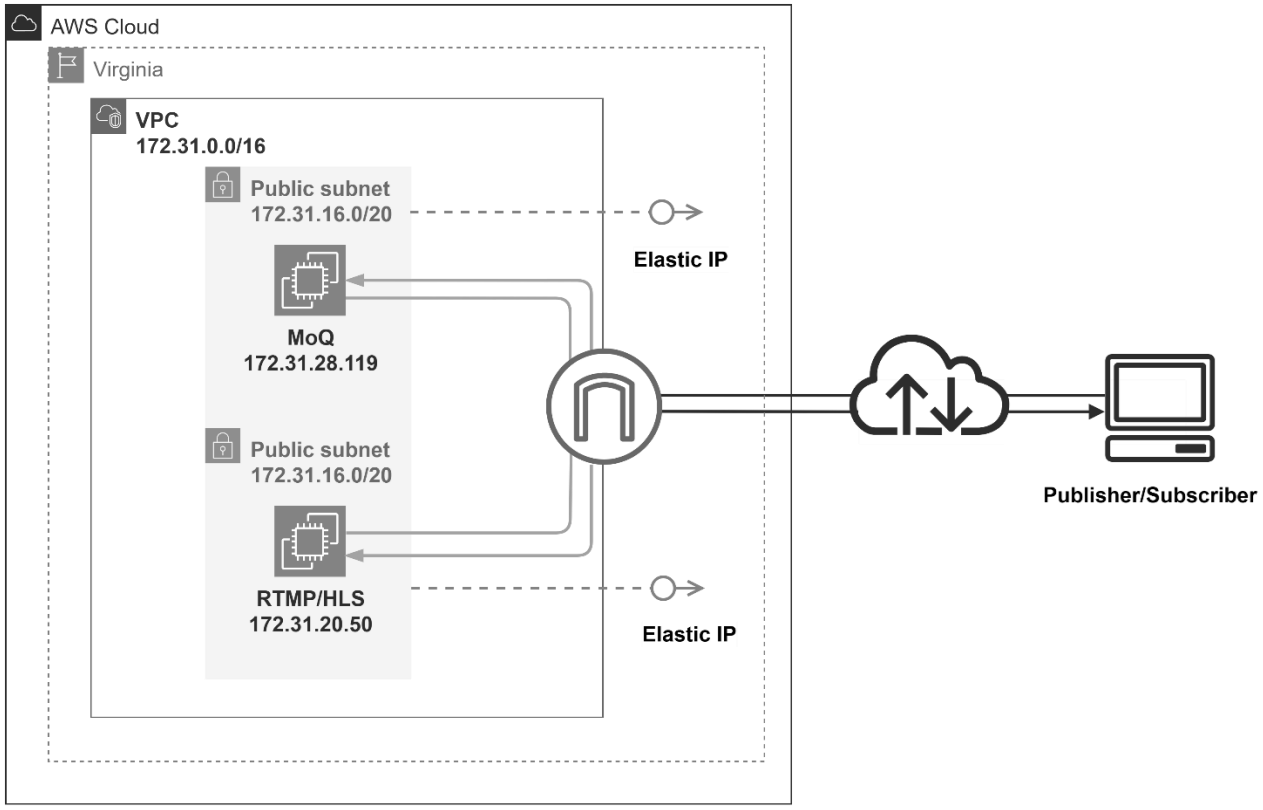
Tx time	Environment	Tests	Video resolution	Bit rate
20 min	Local	12	720p 30 fps	8Mbps

	Remota	12	720p 30 fps	8Mbps
40 min	Local	12	720p 30 fps	8Mbps
	Remota	12	720p 30 fps	8Mbps

Source: Own work

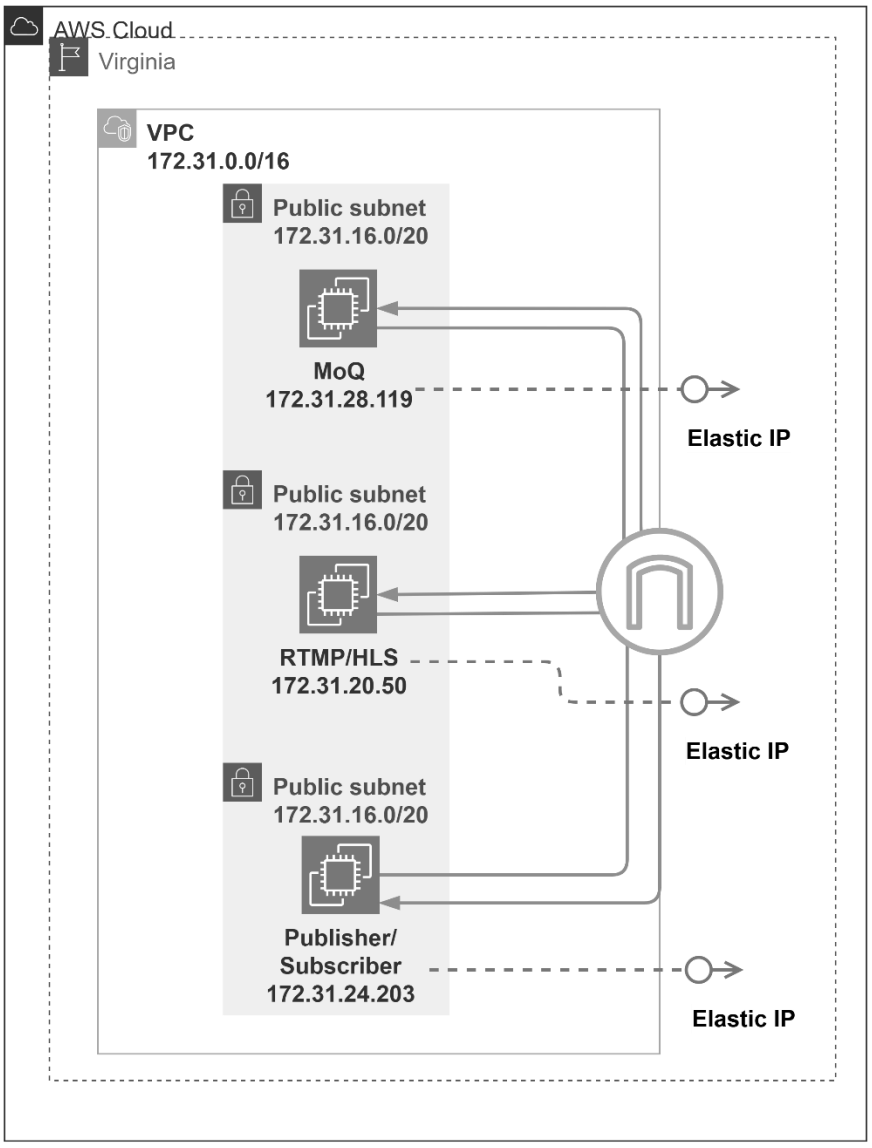
For the tests, a local client is used within the same subnet of the VPC and a remote client over the internet. In the case of MoQ, each client contains a Python script that deploys a web service where the encoding and decoding of the JavaScript-based stream is performed [26]. Figures 6 and 7 show the proposed scenarios, where each instance has an elastic IP to be accessible from the internet.

Figure 6. Remote test scenario



Source: Own work

Figure 7. Test scenario for local client implemented in AWS.



Source: Own work

The clients were configured to make comparisons using the same video resolution and bit rate across all proof-of-concept tests. For the tests, a bit rate of 8Mbps with a resolution of 720p at 30fps is set. For HLS, OBS is used as the video encoder and VLC as the stream decoder. MoQ uses Python to deploy a web service that acts as both the encoder and player for the stream. Finally, Wireshark is used as the traffic analysis tool.

Table 2 specifies the software used for stream transmission and playback on both the local and remote clients, as well as the programs and tools utilized for analyzing the tests performed.

Table 2. Versions and programs used.

Program	**OBS**	**VLC**	**Wireshark**
Version	30.2.2	3.0.21	4.4.0
Tool	**Python**		**tcpdump**
Version	3.12.315		4.99.1

Source: Own work

3. Analysis of Results

Once the performance tests are completed, throughput is defined as the evaluation metric, using Wireshark on the clients and tcpdump on the server. A total of 48 tests are conducted, evaluating the behavior of 24 local streams and 24 remote streams. Half of these tests are conducted with a duration of 20 minutes, while the remaining tests last 40 minutes. Table 3 presents the results obtained using MoQ in each test, including the overall average throughput and its standard deviation.

Table 3. Throughput values and average for MoQ in the tests conducted.

Protocol	**tx duration (min)**	**Environment**	**Execution time**	**Publisher**		**Subscriber**	
				Throughput (Mbps)	**Throughput σ (Mbps)**	**Throughput (Mbps)**	**Throughput σ (Mbps)**

MoQ	20	local	9:30	0,6591	0,0992	0,3841	0,0715
			12:50	0,5764	0,1181	0,2985	0,0855
			21:50	0,5594	0,1789	0,3306	0,1157
		remote	10:30	0,6048	0,1023	0,3502	0,0714
			13:10	0,6065	0,1031	0,6041	0,1111
			22:30	0,6111	0,1022	0,3455	0,8232
	40	local	9:50	0,6681	0,0817	0,3934	0,0528
			13:50	0,5991	0,0704	0,3221	0,0493
			20:20	0,6766	0,1042	0,4027	0,0677
		remote	10:40	0,7005	0,0849	0,4471	0,0791
			14:40	0,6963	0,1016	0,4503	0,0774
			19:30	0,6965	0,0958	0,4536	0,0809
	Average			**0,6379**	**0,1035**	**0,3985**	**0,1405**

Source: Own work

According to the results, it is identified that MoQ has a minimum channel utilization, averaging 0.64 Mbps with an average standard deviation of 0.10 for the publisher. On the subscriber side, the average channel utilization is approximately 0.4 Mbps with a standard deviation of 0.14 Mbps. This indicates a low level of throughput, suggesting that MoQ performs well in environments with limited channel capacity. Table 4 presents the results obtained for HLS in each of the tests, averaging the values and calculating the standard deviation for each one.

Table 4. Throughput values and average for HLS in the tests conducted.

Protocol	tx duration (min)	Environment	Execution time	Publisher		Subscriber	
				Throughput (Mbps)	Throughput σ (Mbps)	Throughput (Mbps)	Throughput σ (Mbps)
HLS	20	local	9:30	8,1846	0,3621	0,9732	0,2062
			12:30	8,1641	0,5288	0,9768	0,1725
			21:30	8,1581	0,5495	1,0021	0,1651
		remote	9:40	8,1098	0,7688	1,0426	0,2002
			13:00	8,1424	0,6337	1,0213	0,167
			22:00	8,1573	0,5408	1,0337	0,4138
	40	local	10:00	8,1751	0,3392	0,8455	0,0883
			12:00	8,2018	0,2878	1,1128	0,1278
			18:40	8,1653	0,5335	1,0761	0,1334
		remote	6:20	8,1943	0,5672	1,0991	0,1377

			15:00	8,2036	0,5621	1,0845	0,1765
			21:10	8,2142	0,4356	1,1291	0,1641
	Average			**8,1726**	**0,5091**	**1,0331**	**0,1794**

Source: Own work

Regarding RTMP + HLS, a higher channel utilization compared to MoQ is observed, with an average of 8.2 Mbps and a standard deviation of 0.5 Mbps for the publisher, and 1.03 Mbps with a standard deviation of 0.17 Mbps for the subscriber. As a final result, it is identified that the protocol operating over QUIC (MoQ) has lower channel utilization compared to RTMP + HLS, which operate over TCP. This is due to how MoQ works, utilizing the publisher/subscriber model through the use of retransmitting servers, which experience much less load since the browser is responsible for encoding and decoding the stream.

Finally, it is observed that, despite previous research on QUIC highlighting how this protocol enables maximum throughput utilization [26] and [27], in this scenario, only the strictly necessary channel utilization is employed to handle video stream delivery to relays. Therefore, MoQ could be used as a protocol in environments with limited throughput, making it suitable for video conferencing or live video streaming. As future work, it is essential to evaluate how MoQ evolves to assess its applications in the industry and its performance in non-ideal network environments.

4. Conclusions

The average obtained in the stream publication for RTMP + HLS (8.17 Mbps) could be related to the bit rate configured in OBS (8 Mbps). This suggests that the channel utilization of HLS is directly affected by the bit rate setting in the encoder, as a higher bit rate leads to increased channel utilization and improved stream quality. The results showed a channel utilization of

8.16 Mbps for RTMP + HLS over TCP and 0.6 Mbps for MoQ in the publication, with 1.04 Mbps and 0.3 Mbps, respectively, for the subscription. This makes MoQ superior to RTMP + HLS due to its lower channel utilization, thanks to the use of retransmitting servers and its publisher/subscriber model.

Based on the throughput results for both protocols, it can be confirmed that HLS has much higher channel utilization than MoQ due to the way each protocol operates. While RTMP relies on the bit rate to define channel utilization, MoQ uses only the necessary bandwidth to publish the stream with the same bit rate and resolution parameters as HLS. Additionally, in both protocols, it is evident that the publication of the stream uses more channel bandwidth, showing a greater difference compared to the subscription of the stream.

Acknowledgments

Product derived from the MAXWELL research seedbed, GISSIC Research Group. Universidad Militar Nueva Granada - Vigencia 2025

References

[1] J. Postel. (1980). User Datagram Protocol. doi.org/10.17487/RFC0768

[2] Defense Advanced Research Projects Agency. (1981). Transmission Control Protocol. doi.org/10.17487/RFC0793

[3] Floyd, S. (2001). A report on recent developments in TCP congestion control. doi:10.1109/35.917508

[4] W. Eddy. (2022). Transmission Control Protocol (TCP). doi.org/10.17487/RFC9293

[5] Iana. (2024). Transmission Control Protocol (TCP) Parameters. https://www.iana.org/assignments/tcp-parameters/tcp-parameters.xml

[6] H. Nielsen, J. Mogul, L. Masinter, R. Fielding, J. Gettys, P. Leach, T. Berners-Lee. (1999). Hypertext Transfer Protocol -- HTTP/1.1

[7] M. Thomson and C. Benfield. (2022). HTTP/2. doi.org/10.17487/RFC9113

[8] K. Seemakhupt and K. Piromsopa. (2016). When should we use HTTP2 multiplexed stream? doi.org/10.1109/JCSSE.2016.7748934

[9] Langley, A., Riddoch, A., Wilk, A., Vicente, A., Krasic, C., Zhang, D., ... & Shi, Z. (2017). The quic transport protocol: Design and internet-scale deployment. doi.org/10.1145/3098822.3098842

[10] Cui, Y., Li, T., Liu, C., Wang, X., & Kuhlewind, M. (2017). Innovating Transport with QUIC: Design Approaches and Research Challenges. doi:10.1109/mic.2017.44

[11] Saurabh, Prakash, S., Singh, P. K., Nandi, S. K., & Nandi, S. (2019). Is QUIC Quicker Than TCP? A Performance Evaluation. doi.org/10.1007/978-3-030-15035-8_12

[12] J. Iyengar, M. Thomson. (2021). QUIC: A UDP-Based Multiplexed and Secure Transport. doi.org/10.17487/RFC9000

[13] Bishop, M. (2022). Hypertext Transfer Protocol Version 3 (HTTP/3). Standards Track. Internet Engineering Task. doi.org/10.17487/RFC9114

[14] Gagliardi, E., Levillain, O. (2020). Analysis of QUIC Session Establishment and Its Implementations. https://doi.org/10.1007/978-3-030-41702-4_11

[15] K. Wolsing, J. Rüth, K. Wehrle, and O. Hohlfeld. (2019). A performance perspective on web optimized protocol stacks: TCP+TLS+HTTP/2 vs. QUIC. doi.org/10.1145/3340301.3341123

[16] H. Ravuri, M. Vega, J. Hooft, T. Wauters and F. De Turck. (2023). Adaptive Partially Reliable Delivery of Immersive Media Over QUIC-HTTP/3. doi: 10.1109/ACCESS.2023.3268008.

[17] Y. Yu, M. Xu and Y. Yang. (2018) When QUIC meets TCP: An experimental study. doi: 10.1109/PCCC.2017.8280429

[18] Sitepu, F. A. V. (2022). Performance Evaluation of Various QUIC Implementations: Performance and Sustainability of QUIC Implementations on the Cloud (Dissertation). urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-92721

[19] L. Curley, K. Pugin, S. Nandakumar, V. Vasiliev and I. Swett. (2024). Media over QUIC Transport. https://datatracker.ietf.org/doc/draft-ietf-moq-transport/05/

[20] Z. Gurel, T. E. Civelek, A. Bodur, S. Bilgin, D. Yeniceri, and A. C. Begen. (2023) Media over QUIC: initial testing, findings and results. doi.org/10.1145/3587819.3593937

[21] Gurel, Z., Civelek, T. E., Ugur, D., Erinc, Y. K., & Begen, A. C. (2024). Media-over-QUIC Transport vs. Low-Latency DASH: a Deathmatch Testbed. doi.org/10.1145/3625468.3652191

[22] V. Vasiliev. (2024). The WebTransport Protocol Framework. datatracker.ietf.org/doc/draft-ietf-webtrans-overview/08/

[23] J. Cenzano. moq-encoder-server. (2024). GitHub repository. Disponible en

github.com/facebookexperimental/moq-go-server

[24] R. Pantos, W. May. (2017). HTTP Live Streaming. doi.org/10.17487/RFC8216

[25] J. Cenzano. lhls-simple-live-platform. (2021). GitHub repository. Disponible en: github.com/jordicenzano/lhls-simple-live-platform

[26] J. Cenzano. moq-encoder-player. (2021). GitHub repository. Disponible en: github.com/facebookexperimental/moq-encoder-player

[27] H. K. Ravuri, M. T. Vega, J. D. Van Hooft, T. Wauters and F. De Turck. Adaptive Partially Reliable Delivery of Immersive Media Over QUIC-HTTP/3. 2023. doi: 10.1109/ACCESS.2023.3268008.