Streaming over QUIC: Applications in Media and IoT

Archit Sengupta

April, 2025

Abstract

In this study, I explore the capabilities and applications of the QUIC protocol in real-time media streaming and IoT data transmission. The work is divided into two sections. The first focuses on video streaming over QUIC. It highlights progress in the IETF and related industry adaptations. The second looks at the integration of QUIC with IoT protocols. This includes a custom implementation for streaming IMU data to an Azure server and performance comparisons against TCP.

Introduction

QUIC (Quick UDP Internet Connections) is a modern transport protocol originally developed by Google and then standardized by the IETF. It was designed to overcome the limitations of TCP. QUIC operates over UDP and offers stream multiplexing, stream prioritization, connection migration, 0-RTT handshakes, and mandatory encryption. These features make it really good for applications which require low-latency like real-time media streaming and IoT/sensor data transfer.

QUIC's development culminated in several key RFCs, most notably:

- RFC 8999 – Version-Independent Properties of QUIC
- RFC 9000 – QUIC: A UDP-Based Multiplexed and Secure Transport
- RFC 9001 – Using TLS to Secure QUIC
- RFC 9002 – QUIC Loss Detection and Congestion Control

The issue with TCP is that it suffers from head-of-line blocking at the transport layer. QUIC supports independent, concurrent streams over the same connection, therefore any congestion on 1 stream doesn't affect the other. This is amazing for our considered use cases, where multiple streams with different priority levels can run at the same time.

Another crucial benefit is that QUIC completes the TLS handshake in the 1st round trip itself, thereby establishing a connection in 1 RTT. Optimizations with session key storage can reduce this to 0 RTT as well, reducing connection establishment time greatly.

In this paper I look at two promising applications of QUIC: streaming real-time media and transmitting IoT data. Both contexts benefit from QUIC's core innovations. I conduct a literature survey, looking through the studies and research conducted till date, and progress being made. I look at current draft protocols and include results from my own implementation and experiments.

Section 1: Media Streaming Over QUIC

Motivation

Media delivery is a big part of the internet experience today and protocols that can handle low latency and reliability are vital. QUIC is a transport protocol developed by Google and adopted by the IETF. It is designed for speed, security, and stream multiplexing. In media streaming, QUIC allows for flexible prioritization of data streams and integrates security by default.

Consider the current ecosystem, we have 3 different requirements:

- **Ultra-low latency (under 200 ms)**: Used in applications like video conferencing and calls. These are usually implemented using WebRTC over UDP peer-to-peer. This allows very low latency but it's not scalable when many users are involved.
- **Medium latency (1-5 seconds)**: Common in video-on-demand and OTT streaming platforms. A few seconds of latency is acceptable here, and several mature protocols and systems exist for these use cases (HLS, DASH, RTMP).

- **Low latency (200 ms - 1 second)**: Used in scenarios like sports live streams and interactive online gaming. These currently rely on solutions such as Low Latency DASH and Low Latency HLS, which are built on top of TCP. However, TCP limits their potential due to head-of-line blocking and lack of prioritization. This is the prime target for Media over QUIC Transport.

Background and Related Work

The field of media streaming over QUIC has seen a lot of research across several domains. I summarize key contributions below:

QUIC as a Transport for Video Streaming

The first study which studied QUIC as a transport protocol for video streaming was *Advanced Transport Options for DASH-Based Streaming* [src] [June, 2016] which assesses DASH over QUIC. It highlighted how QUIC reduces head-of-line blocking and improves startup delay. They tested multiple protocols including HTTP/1.1 and SPDY over QUIC, finding that while QUIC had slightly higher overhead, it maintained link utilization above 80% even under high RTTs. It demonstrated better responsiveness than HTTP/2 under network fluctuation conditions.

Selective Reliability and Stream-Level Control

A notable innovation is the use of unreliable streams in QUIC to send non-critical video frames (like B-frames), reserving reliable delivery for keyframes. This approach, demonstrated in *The QUIC Fix for Optimal Video Streaming* [src] [Sep 2018] provides selective reliability using datagrams. This reduces latency while preserving video coherence. The integration of FEC techniques further mitigates the impact of loss. However these kinds of unreliable datagrams have not been given much attention in standard implementations.

Adaptive Bitrate Streaming with Retransmissions

Multiple works have proposed adaptive bitrate (ABR) schemes that utilize QUIC's retransmission flexibility. Techniques like SQUAD discussed in *Improving QoE of ABR Streaming with QUIC Retransmissions* [src] [Oct, 2018] utilize QUIC to retransmit higher-quality segments (for parts which haven't been played yet) when bandwidth improves. This reduces video quality switches and increases overall average bitrate. Real-world experiments showed that QUIC with retransmissions, delivered higher QoE compared to TCP and HTTP/2 in high latency/loss networks.

In *QUICker or not? - An Empirical Analysis of QUIC vs TCP for Video Streaming QoE Provisioning* [src] [Feb, 2019], the authors compared video streaming over QUIC as compared to TCP. They did not find any improvements using QUIC, but I judge their study seems to be flawed since they simply analyzed systems like Youtube. They would stream directly to browsers which might not implement custom which might not add custom application level prioritization/ multiplexing within the streams.

Scalable High Efficiency Video Coding based HTTP Adaptive Streaming over QUIC Using Retransmission [src] [August, 2020] showed that retransmitting segments with higher quality using QUIC improves overall video quality in SVC formats without increasing latency too much. The use of scalable video coding (SVC) layers over QUIC lets clients recover lost enhancement layers when bandwidth becomes available. This is not feasible over TCP due to retransmission delays.

Another recent work, *Days of Future Past* [2021] src, introduced a new adaptive bitrate algorithm that predicts future bandwidth conditions using a feedback loop. This lets the streaming client preemptively request higher-bitrate content when the future capacity is forecasted. The algorithm makes use of the fast recovery and low-latency characteristics of QUIC. This makes delivery consistent and gives better playback stability when network conditions are fluctuating.

Media Over QUIC Transport(MoQT) and Streaming Protocol Evolution

IETF MoQT

Realizing the need for a standardized protocol for streaming video over QUIC the IETF created a working group which would implement this next generation of streaming. The protocol was named Media over QUIC Transport (MoQT) [src]. The main motivation is to use QUIC for 'large-scale media transmission in one-to-one, one-to-many, and many-to-one applications that might require interactivity'. They want to standardize a low-latency, scalable and QUIC-native transport for media delivery. The core MoQT draft proposes a publish-subscribe framework where media is structured as tracks, groups, and objects. Each of

these is independently transmitted over multiplexed QUIC streams. This model means greater control over the stream prioritization and timing.

- *Need for Low-Latency Media over QUIC*[src][June, 2023] provides the foundations and reasoning for MoQT. It identifies use cases where current HTTP-based methods fall short, such as real-time gaming, remote collaboration and live streaming. It makes the case for a more responsive transport architecture capable of subsecond end-to-end latency.
- In '*Media over QUIC: Initial Testing, Findings, and Results*' [src][2023], researchers benchmarked MoQT implementations and evaluated performance under different network conditions. They found that QUIC's built-in transport-level reliability and stream separation could reduce startup time, jitter, and packet loss, especially when relays and publishers were correctly tuned.
- Building on these foundations, the *Design and Implementation of a Low-Latency Origin and Relay for Media-over-QUIC Transport* [src] [April, 2024] presents a reference implementation of a full MoQT delivery system. Relays are treated as first-class entities and prioritization at the network level is shown to improve latency and delivery reliability. Studies tested relay performance with various prioritization methods, including First-Come-First-Served and Last-Come-First-Served, to evaluate quality-latency trade-offs.

These findings collectively confirm that QUIC is not just a faster TCP replacement. It's a versatile transport for next-generation streaming systems requiring real-time data transfer.

WARP Protocol

Twitch introduced the WARP protocol which is a variant tailored to live and interactive streaming on their platform. This is also currently an IETF draft [src] WARP builds on MoQT's core architecture but introduces specific mechanisms like:

- subscription channels which are controlled by the client
- Relay-level object prioritization
- Failover-aware segment fetching

The WARP architecture is designed to provide low-latency, fragmented CMAF segments to millions of concurrent viewers while maintaining interactivity.

MoqTransfork

The MoqTransfork draft [src] by Luke Curley proposes extensions to MoQT to allow for greater flexibility of transport. It adds new session-level constructs, such as:

- Fetch and subscribe types
- Granular filtering capabilities
- Support for conditional retrieval

These extensions are designed to support use cases beyond live media, including archived playback, on-demand object fetching, and content-aware routing.

These drafts and research papers show a move toward establishing a QUIC-native media transport ecosystem. MoQT introduces application-layer protocols designed specifically for media delivery over QUIC.

Prioritization Strategies

One of QUIC's most useful features is the ability to assign priorities to individual streams. For example, I could prioritize audio over video to reduce user discomfort during network degradation. Further, for live video, there's a key decision to be made: should I prioritize more recent frames or preserve temporal order even if the viewer is lagging? The answer would most probably be some kind of middle ground.

The paper *Prioritization in Media over QUIC Transport* [src][Feb, 2024] discusses how QUIC's stream prioritization APIs can be applied at the media application layer. Through several experiments, it shows dynamic stream reprioritization for video layers or separation of audio and video. It discusses 2 different strategies:

- **Implicit Prioritization (FEFS)**: A single unidirectional QUIC stream is used to deliver the frames in the encoded order. This is also known as First Encode, First Send (FEFS). When there is not enough bandwidth to send the encoded media, a latency increase may occur.

Figure 1: FEFS

- **Prioritization by Frame Type (FT)**: A separate unidirectional QUIC stream is created for each frame type (i.e., I, P and B) as shown in the figure, and based on the type, each stream is assigned a priority. I and B-frames have the highest and lowest priority, respectively.

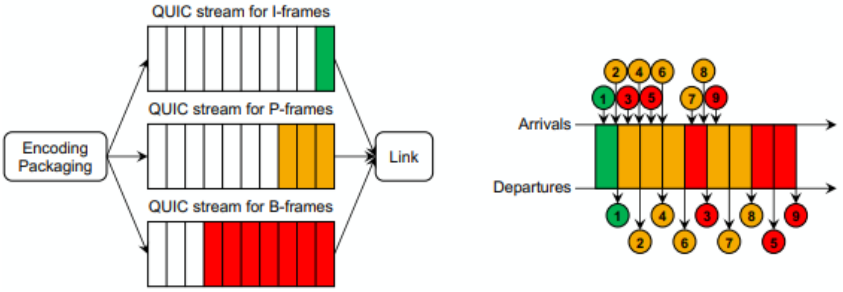


Figure 2: FT

My current streaming server and client implementation (under development) [QUIC-Streaming], is being designed to test the trade-offs using multiple QUIC streams and prioritization logic.

Investigating Fingerprinting Attacks on QUIC Media Streaming

I investigated whether fingerprinting attacks are possible on encrypted media streaming traffic using the QUIC protocol. Despite QUIC's encryption I found that side-channel metadata such as bitrate, packet burst size, and timing still leaks information. My goal was to see whether it's possible to infer the resolution of a video stream from encrypted QUIC traffic.

I conducted experiments in two different settings and captured packets of video streams using Wireshark:

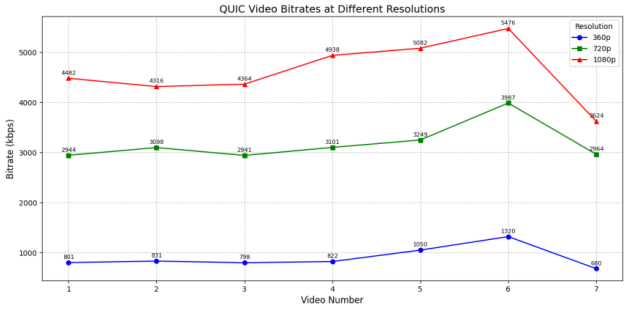
- **Controlled MoQT Testbed**: I set up a local test environment using a modified MoQT (Media over QUIC Transport) demo [TugasAkhir-QUIC/public-moq-demo]. I collected data points from multiple different kinds of videos and found the average bitrate. Using this I attempted to predict the video resolution for a test set of 15 samples. I established tighter lower and upper bounds for the three video qualities.

  - 360p: bitrate $\leq$ 1500 kbps
  - 720p: 2500 kbps $\leq$ bitrate $\leq$ 4000 kbps
  - 1080p: bitrate $\geq$ 4000 kbps

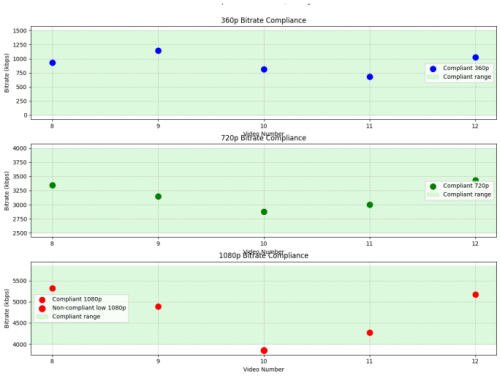
  I achieved 93% accuracy in predicting video resolution. The bitrate differential was very consistent across every resolution which made prediction easier.

Table 1: Training Data for Local Video Bitrates by Resolution (in kbps)

Video	360p (kbps)	720p (kbps)	1080p (kbps)
1	801	2944	4482
2	831	3098	4316
3	798	2941	4364
4	822	3101	4938
5	1050	3249	5082
6	1320	3987	5476
7	680	2964	3624



(b) Experimental testbed setup



(c) Local test data compliance

Figure 3: Local Testbed Video Quality prediction results

- **Real-World YouTube Streaming**: Youtube implements several countermeasures to prevent fingerprinting. This includes randomized segment delivery and protocol switching. I captured QUIC and UDP traffic during YouTube sessions and discovered that the security implementations did reduce fingerprinting accuracy. However, by focusing on burst sizes rather than raw bitrate, I was still able to uncover patterns linked to video quality. Focusing on total bytes sent per burst, we established these bounds:
  - 360p: 600,000 - 1,300,000 bytes (median: $\approx 870,000$ bytes)
  - 1080p: 1,700,000 - 4,300,000 bytes (median: $\approx 2,700,000$ bytes)
  - 4K: $\geq 19,000,000$ bytes (range: 19,000,000 - 39,000,000 bytes)

  This resulted in an accuracy of 77%.

Table 2: Youtube QUIC Media Streaming Statistics

Resolution	Total_avg_bitrate	Avg_bitrate_for_bursts	Total_bytes_sent_per_burst
144p	4220	15402.0000	534212.00
144p	4894	12463.0000	444905.00
144p	994	25902.0000	652442.00
144p	582.6	15531.0000	1233403.00
360p	1524.19	12406.7317	611093.20
360p	1984	44777.0000	1210815.00
360p	6503	63197.0000	913640.00
360p	1840.21	16203.5350	870151.40
720p	4584	40415.0000	2520136.00
720p	8260	94583.0000	2065680.00
720p	10165	18034.0000	1350812.00
720p	10136	25068.0000	1401037.00
1080p	17612.7	45526.0000	2688331.00
1080p	5900	59682.0000	4284151.00
1080p	4551.1	18937.7318	1682338.00
1080p	7458.8	20642.4499	2795837.20
4k	22386.98	53157.5039	21835676.00
4k	28421	203197.0000	19879211.00
4k	28196	76225.0000	20682550.00
4k	38752	27157.0000	38566643.00
4k	93779	215829.0000	20659215.00

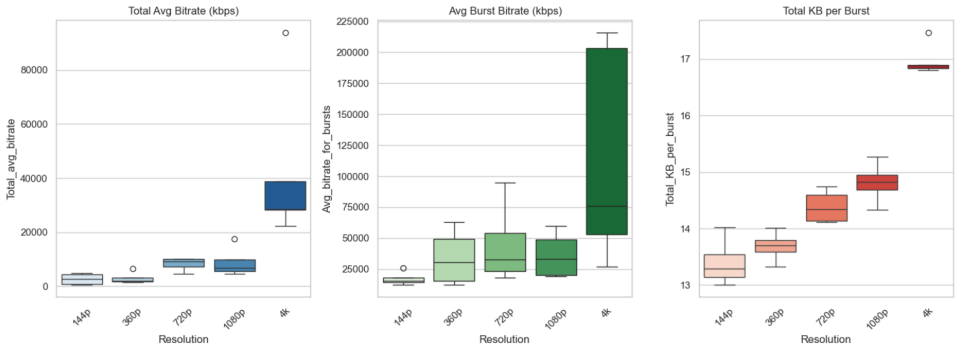


Figure 4: Burst patterns across resolutions

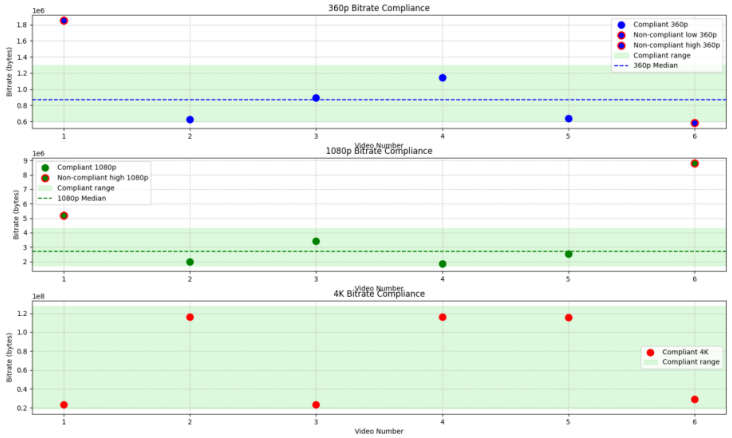


Figure 5: Prediction and Rule Compliance

This work builds on previous research in side-channel attacks. For example, Reed and Kranch demonstrated fingerprinting of Netflix videos over HTTPS [33], while Zhao et al. showed that video resolution could still be inferred from encrypted QUIC streams [34]. Their methods inspired me to adapt traditional fingerprinting techniques to the more complex environment of QUIC and MoQT.

Key findings:

- In controlled environments, QUIC streams leak enough information to infer video resolution with high accuracy.
- YouTube's deployment introduces noise and variability, but burst size analysis still gives inference capabilities.
- QUIC's encryption does not fully protect against passive traffic analysis.

Streaming IoT Data Over QUIC

Motivation

IoT data transmission requires low latency, efficiency, and security. Application level protocols like MQTT and AMQP are widely used, but they rely on TCP or UDP. With QUIC's features, 0-RTT handshakes, multiplexed streams, and native encryption, the ecosystem can be improved a lot in IoT.

Background and Related Work

Foundational Studies

The paper *Towards Securing the Internet of Things with QUIC* [src] (Eggert, 2020) provided empirical evaluation of QUIC on resource-constrained IoT devices such as the Particle Argon and ESP32-DevKitC V4. It quantified the protocol's resource requirements. The study gave optimizations such as disabling connection migration and using hardware cryptography to reduce overhead. It said that QUIC is good for IoT devices with at least 32 KB RAM and 100 KB of storage. It established a baseline for low-power implementations.

MQTT over QUIC

MQTT (Message Queuing Telemetry Transport) is a lightweight, publish-subscribe network protocol designed for constrained devices and low-bandwidth, high-latency, or unreliable networks. It is efficient and allows real-time communication between clients through a central broker. A fair bit of work has explored the integration of MQTT with QUIC to address performance limitations in traditional TCP/TLS-based MQTT systems.

1. In *Implementation and Analysis of QUIC for MQTT* (Kumar & Dezfouli, 2019) [src], it was shown that MQTT over QUIC improved delivery latency, connection overhead, and resource usage on IoT hardware. The implementation had lightweight QUIC agents, redesigned data structures for efficient use on constrained devices. The benefits included:
    - 56.2% fewer packets during connection establishment compared to MQTT over TCP
    - 55.6% reduction in message delivery latency under packet loss
    - Up to 83.2% lower processor usage and 50.3% lower memory usage in half-open connection scenarios
    - Better resilience during connection migration
2. More recent work by Fernández et al. (2020) *And QUIC meets IoT: performance assessment of MQTT over QUIC* [src], evaluated MQTT over QUIC in ns-3 simulations with WiFi, 4G/LTE, and satellite conditions. It used real MQTT clients and brokers. The study showed
    - Up to 40% reduction in publish/subscribe message completion time on WiFi at 5% packet loss
    - Better performance stability in lossy networks than TCP

- Improvements in connection setup time for MQTT connections

3. Jeddou et al. (2022) *Delay and Energy Consumption of MQTT over QUIC: An Empirical Characterization Using Commercial-Off-The-Shelf Devices* [src] conducted an evaluation of MQTT over QUIC on commercial off-the-shelf Raspberry Pi hardware. Their findings showed:

   - QUIC achieved lower average one-way delays than TCP over WiFi, cellular, and satellite networks
   - Jitter was reduced by 41–61% under 5% packet loss
   - QUIC required fewer retransmissions at increasing packet loss rates, with comparable energy consumption to TCP

   These studies confirm that MQTT over QUIC increases performance and reliability and remains energy efficient. This is critical factor for battery-powered IoT and IIoT (Industrial IoT).

4. The emergence of EMQX 5.0 in 2022 [src] marked the first production-grade MQTT-over-QUIC broker. It supports:

   - 0-RTT handshakes for rapid reconnections
   - Fallback to TCP if QUIC fails
   - 30–50% lower latency in high-latency networks (eg:, satellite or cellular)

AMQP over QUIC

AMQP (Advanced Message Queuing Protocol) is a secure, reliable, and standardized messaging protocol that lets systems to communicate through message brokers with features like separated message queuing, routing, and transactions. It is designed for more complex, enterprise-level messaging needs where guaranteed delivery and security are important. Recent research by Iqbal et al. (April, 2023) across two papers:

- *Use of QUIC for AMQP in IoT networks* [src]
- *Performance evaluation of AMQP over QUIC in the internet-of-thing networks* [src]

has explored the integration of the Advanced Message Queuing Protocol (AMQP) 1.0 with QUIC for IoT environments. Their study gave evidence that AMQP over QUIC has lower latency, improved energy efficiency, and strong resilience in unstable networks compared to implementation over TCP.

The researchers evaluated AMQP 1.0 over QUIC and TCP ssing the NS3 simulator and Docker-based containerr. It was tested across WiFi, 4G/LTE, and satellite networks. Performance metrics included startup latency, total communication time, energy consumption, and resilience to packet loss.

- Startup Latency was reduced by 52% (WiFi), 38% (4G/LTE), and 34% (Satellite) when using QUIC.
- Communication Time was lowered by up to 8.57% on satellite networks, showing QUIC's edge in difficult conditions.
- Energy Efficiency: QUIC reduced energy usage by 31% during tests involving multiple short connections, making it highly suitable for battery-powered devices.
- Packet Loss Resilience: At 15% loss rates, QUIC's performance degradation was 4–9% depending on network type, compared to 16–36% for TCP.
- Data Throughput: QUIC transmitted 3.5 times more data than TCP, yet achieved 7 times higher throughput due to more efficient communication patterns.

TCP performed slightly better under extremely low-bandwidth conditions, but QUIC outperformed TCP in nearly all other scenarios. These results show QUIC can be used as the transport layer for AMQP as well. It again offers benefits in latency and energy efficiency in real-world IoT deployments.

Specialized Use Cases

- **Industrial IoT (IIoT):** Pérez-Díaz et al. [src] (2021) demonstrated reliable telemetry transmission in mining environments despite intermittent 4G/WiFi handoffs.
- **Internet of Vehicles (IoV):** EMQX 5.0 benchmarks confirmed that QUIC maintained stable connections across transitions from 4G to WiFi, reducing reconnection latency by up to 70%. [src] [src]

Custom Implementation: IMU Data Streaming Over QUIC

To evaluate QUIC for real-time sensor data, I built a system that streams data from an IMU (Inertial Measurement Unit) over QUIC using `python-aioquic` and compared results.

Methodology

The IMU data stream includes readings from both the accelerometer and gyroscope, with updates generated at high frequency to mimic realistic real-time sensor behavior.

Two transport layers were implemented:

- A QUIC-based client-server system using `aioquic`
- A TCP-based client-server system using Python's standard libraries for comparison

Both client-server pairs were deployed on devices that streamed data to a remote server hosted on Microsoft Azure (West US region). The network latency, packet loss, and jitter between the sender and receiver were left unoptimized to simulate real-world constraints.

Streaming Modes Tested

I tested three distinct data transmission modes using the QUIC protocol:

1. **Single Stream Mode:** All sensor data was serialized and transmitted over a single QUIC stream. This mode provides a baseline and tests QUIC's performance in a straightforward configuration.
2. **Multi-stream Mode (No Prioritization):** Separate QUIC streams were created for each sensor (one for accelerometer, one for gyroscope), and data was transmitted in a FIFO (First-In-First-Out) order with no prioritization.
3. **Multi-stream Mode (With Custom Prioritization):** In this mode, a custom prioritization mechanism was added to the QUIC client. Since `aioquic` does not natively support prioritization, I implemented a scheduling algorithm that tracks the total number of bytes sent per stream. At each decision point, the client selects the stream to transmit on based on weighted priorities assigned to each sensor stream.

For this test, accelerometer data was given higher priority than gyroscope data by assigning it a larger weight in the scheduler.

Motivation for Prioritization

In real-time applications with low bandwidth or unstable network, prioritization makes sure that the most relevant or time-critical data is delivered with minimal delay and packet loss. This can be useful for several use cases. For example :

- **Body Movement Tracking for Biomechanics or VR:** In systems where multiple sensors are placed across the body, data from critical joints (knees, elbows) or the head may be prioritized to always show posture or orientation.
- **Robotics and Human-Robot Interaction:** A robot equipped with multiple IMUs on different limbs may prioritize data from the arms or ends of the limbs. This is because during tasks like gesturing or movement those parts interact directly with the environment.
- **Aerospace and UAV Stability Systems:** In drones or unmanned aerial vehicles (UAVs), IMUs provide crucial orientation and stability data. During flight sensor data related to pitch, yaw and roll is important for maintaining control. Prioritizing these readings over other sensor data can make sure that critical updates for flight are received first.

These use cases benefit from dynamic prioritization, where sensor importance can be updated based on context or activity type.

Measurement and Evaluation

To evaluate the performance of each mode, I measured the rate of delivery of each sensor's data at the receiver side. This included the number of updates received per second for each sensor stream. By comparing these delivery rates across the different streaming modes, as well as comparing QUIC with TCP, I was able to deduce how well QUIC performed as compared to TCP.

Results and Analysis

Duration (s)	Accel Rate (msgs/sec)	Gyro Rate (msgs/sec)
5.00	250.33	250.32
10.00	260.60	260.60
15.01	252.44	252.44
20.01	247.26	247.26
25.01	243.88	243.88
30.01	243.65	243.65
35.02	241.24	241.24
40.02	239.78	239.78
45.02	240.73	240.73

Table 3: Message rates in QUIC single stream configuration.

QUIC with Single Stream This configuration uses a single QUIC stream for both sensors and achieved a lower throughput (240.73 msgs/sec for both sensors). This may be due to limitations in the `aioquic` Python library or inefficiencies in sharing one stream. Still, the message rate remained steady over time, making this setup suitable for lightweight or resource-constrained applications.

Duration (s)	Accel Rate (msgs/sec)	Gyro Rate (msgs/sec)
5.00	390.77	390.79
10.00	367.54	367.56
15.00	365.55	365.55
20.01	360.39	360.39
25.01	363.26	363.27
30.01	364.26	364.27
35.01	366.65	366.65
40.02	366.69	366.69
45.02	364.72	364.72

Table 4: Message rates with QUIC and no prioritization.

QUIC, Multiple streams with No Prioritization With multiple streams but no prioritization logic, QUIC handled both streams equally. The system achieved 364.72 msgs/sec, about 8% lower than the prioritized configuration. While this setup delivers strong throughput, it lacks optimization for real-time applications where one data stream may be more critical than the other.

Duration (s)	Accel Rate (msgs/sec)	Gyro Rate (msgs/sec)
5.00	324.34	324.32
10.00	370.77	370.75
15.00	375.83	375.82
20.00	377.16	377.15
25.00	371.48	371.48
30.01	378.12	378.13
35.01	385.95	385.96
40.01	392.88	392.88
45.01	395.54	395.53

Table 5: Message rates with QUIC and prioritized streams.

QUIC with Prioritized Streams This configuration used two streams, prioritizing accelerometer data. It achieved the highest throughput—395.54 msgs/sec (accelerometer) and 395.53 msgs/sec (gyroscope). Although both sensors send data at the same rate, prioritized delivery allowed accelerometer data to arrive slightly earlier than gyroscope data. Over time, rates equalized. This approach is ideal for low-latency, high-reliability applications where one data stream must be prioritized.

Duration (s)	Accel Rate (msgs/sec)	Gyro Rate (msgs/sec)
6.0	172.37	172.33
11.0	201.25	201.01
16.0	166.99	166.93
21.0	254.08	254.03
26.0	307.84	307.80
31.1	344.17	344.13
36.1	370.26	370.24
41.1	351.42	351.38
46.1	340.50	340.46

Table 6: Message rates using TCP transport.

TCP Transport TCP produced relatively high message rates—340.50 msgs/sec (accelerometer) and 340.46 msgs/sec (gyroscope). However, the rates fluctuated more than QUIC-based configurations. This inconsistency reduces suitability for real-time tasks. TCP may work better in buffered or short data bursts where consistent latency isn't critical.

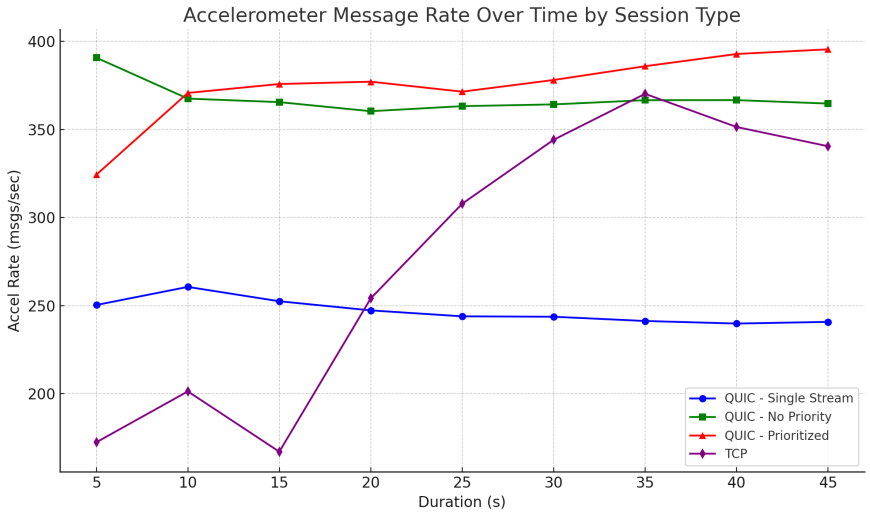


Figure 6: Message rates over time across sessions

Session Type	**Average Accel Rate (msgs/sec)**	**Average Gyro Rate (msgs/sec)**
QUIC - Single Stream	240.73	240.73
QUIC - No Priority	364.72	364.72
QUIC - Prioritized	395.54	395.53
TCP	340.50	340.46

Table 7: Summary of average message rates across different session types.

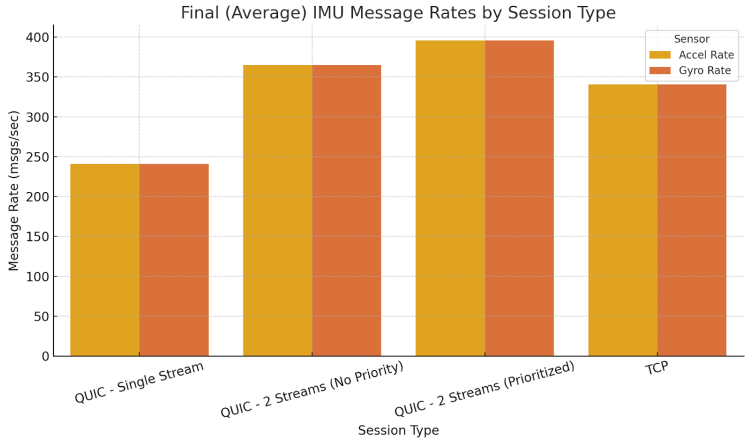


Figure 7: Average message rates

Message rates varied in performance and consistency across different transport configurations. The differences in stream handling and protocol behavior affected delivery patterns a lot. The results show how the QUIC protocol and stream structures can improve raw throughput and stability over time.

References

[1] RFC 8999 – Version-Independent Properties of QUIC. `https://www.rfc-editor.org/rfc/rfc8999`

[2] RFC 9000 – QUIC: A UDP-Based Multiplexed and Secure Transport. https://www.rfc-editor.org/rfc/rfc9000

[3] RFC 9001 – Using TLS to Secure QUIC. https://www.rfc-editor.org/rfc/rfc9001

[4] RFC 9002 – QUIC Loss Detection and Congestion Control. https://www.rfc-editor.org/rfc/rfc9002

[5] Timmerer, C., & Bertoni, A. (2016). Advanced Transport Options for DASH-Based Streaming. *arXiv preprint*. https://arxiv.org/pdf/1606.00264

[6] Ghobadi, M., & Wetherall, D. (2018). The QUIC Fix for Optimal Video Streaming. *IEEE Conference*. https://www.researchgate.net/publication/327930175_The_QUIC_Fix_for_Optimal_Video_Streaming

[7] Bentaleb, A., et al. (2018). Improving QoE of ABR Streaming with QUIC Retransmissions. *ACM Multimedia*. https://dl.acm.org/doi/10.1145/3240508.3240664

[8] Casas, P., et al. (2019). QUICker or Not? An Empirical Analysis of QUIC vs TCP for Video Streaming QoE Provisioning. *IEEE Conference*. https://ieeexplore.ieee.org/document/8685913

[9] Kumar, S., et al. (2020). Scalable High Efficiency Video Coding Based HTTP Adaptive Streaming over QUIC Using Retransmission. *Sensors*. https://www.researchgate.net/publication/341679010

[10] Lorenzi, D., Nguyen, M., Tashtarian, F., Milani, S., Hellwagner, H., & Timmerer, C. (2021). *Days of future past: an optimization-based adaptive bitrate algorithm over HTTP/3*. https://dl.acm.org/doi/10.1145/3488660.3493802

[11] Kumar, P., & Dezfouli, B. (2019). *Implementation and Analysis of QUIC for MQTT*. https://www.sciencedirect.com/science/article/abs/pii/S1389128618310776

[12] De Coninck, Q., et al. (2020). When QUIC Meets IoT: Performance Assessment of MQTT over QUIC. *IEEE IoT Journal*. https://ieeexplore.ieee.org/document/9253384

[13] Pérez-Díaz, N., et al. (2022). Delay and Energy Consumption of MQTT over QUIC: An Empirical Characterization. *Sensors*. https://www.mdpi.com/1424-8220/22/10/3694

[14] Curley, L., et al. (2023). Prioritization in Media over QUIC Transport. *ACM SIGCOMM*. https://dl.acm.org/doi/pdf/10.1145/3638036.3640280

[15] EMQX. (2022). Get Started with MQTT over QUIC. *EMQX Blog*. https://dev.to/emqx/get-started-with-mqtt-over-quic-a-quick-guide-for-the-next-generation-iot-standard-protocol-41nn

[16] Curley, L. (2023). The MoqTransfork Draft. *IETF Draft*. https://datatracker.ietf.org/doc/draft-lcurley-moq-transfork/

[17] Pérez-Díaz, N., et al. (2021). Even Lower Latency in IIoT: Evaluation of QUIC in Industrial IoT Scenarios. *PubMed*. https://pubmed.ncbi.nlm.nih.gov/34502627/

[18] Curley, L. (2023). WARP: Web Transport Protocol. *IETF Draft*. https://datatracker.ietf.org/doc/draft-lcurley-warp/04/

[19] Curley, L., et al. (2023). Media over QUIC: Initial Testing, Findings, and Results. *ACM SIGCOMM*. https://dl.acm.org/doi/10.1145/3587819.3593937

[20] IETF Media over QUIC (moq) Working Group Charter. https://datatracker.ietf.org/group/moq/about/

[21] Curley, L., Pugin, K., Nandakumar, S., Vasiliev, V., Swett, I., et al. Media over QUIC Transport (MOQT), IETF Internet-Draft, 2025. https://datatracker.ietf.org/doc/draft-ietf-moq-transport/

[22] Media over QUIC Transport (MOQT) – Latest Draft.
https://moq-wg.github.io/moq-transport/draft-ietf-moq-transport.html

[23] Streaming Media. Media Over QUIC and the Future of High-Quality, Low-Latency Streaming, Dec. 2024.
https://www.streamingmedia.com/Articles/Editorial/Featured-Articles/Media-Over-QUIC-and-the-Future-of-High-Quality-Low-Latency-Streaming-167165.aspx

[24] IETF Blog. What's the deal with Media Over QUIC?, Jan. 2024.
https://www.ietf.org/blog/moq-overview/

[25] Nanocosmos Blog. Media Over QUIC (MoQ) Explained: Benefits, Use Cases, and How It Works, Jan. 2025.
https://www.nanocosmos.de/blog/media-over-quic-moq/

[26] Gurel, Z., Erkilic Civelek, T., & Begen, A. C.
Need for Low Latency: Media over QUIC. In *Mile-High Video Conference (MHV '23)*, May 2023.
https://www.researchgate.net/publication/371668070_Need_for_Low_Latency_Media_over_QUIC

[27] Nguyen, M., Timmerer, C., Pham, S., Silhavy, D., & Begen, A. C.
Design and Implementation of a Low-Latency Origin and Relay for Media-over-QUIC Transport. In *Proceedings of the 15th ACM Multimedia Systems Conference*, 2024.
https://dl.acm.org/doi/10.1145/3625468.3652914

[28] Künnemann, R., & Steel, G.
Towards Securing the Internet of Things with QUIC. In *NDSS Symposium*, 2020.
https://www.ndss-symposium.org/wp-content/uploads/2020/04/diss2020-23001-paper.pdf

[29] Faheem, I., Moneeb, G., Hanen, K., Walid, K., Seok-Joo, K., & Jin-Ghoo, C.
Use of QUIC for AMQP in IoT networks. *Computer Networks*, 2023.
https://www.sciencedirect.com/science/article/pii/S1389128623000853

[30] Faheem, I., Moneeb, G., Alquhayz, H., Seok-Joo, K., & Jin-Ghoo, C.
Performance evaluation of AMQP over QUIC in the internet-of-thing networks. *Journal of King Saud University - Computer and Information Sciences*, 2023.
https://www.sciencedirect.com/science/article/pii/S1319157823000551

[31] EMQX. MQTT over QUIC: Next-Generation IoT Standard Protocol, 2022.
https://www.emqx.com/en/blog/mqtt-over-quic

[32] EMQX. Will QUIC Become the Next-Generation IoT Protocol? Weekly Ep12, 2022.
https://emqx.substack.com/p/weekly-ep12-will-quic-become-the

[33] A. Reed and M. Kranch. *Identifying HTTPS-Protected Netflix Videos in Real-Time*, 2015. Available: source

[34] Y. Zhao et al. *Identifying Video Resolution from Encrypted QUIC Streams in Segment-combined Transmission Scenarios*, NOSSDAV '24, ACM. Available: https://doi.org/10.1145/3651863.3651883

[35] L. Raji et al. *Tunneling QUIC Through QUIC: On the Feasibility of QUIC-Based Website Fingerprinting Defenses*, PETS 2024. Available: https://petsymposium.org/popets/2024/popets-2024-0112.pdf