Multipath Media-over-QUIC (MoQ) for Video Conferencing Applications

Zuji Zhou

Delft University of Technology

TU Delft

Multipath Media-over-QUIC (MoQ) for Video Conferencing Applications

by

Zuji Zhou

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday May 12, 2026 at 10:30.

Thesis Advisor:	Prof. Fernando Kuipers
Supervisor:	Dr. Nitinder Mohan
	Dr. Tanya Shreedhar
Project Duration:	July, 2025 - May, 2026
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

TU Delft

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Delft, Monday 4th May, 2026 Zuji Zhou

Acknowledgements

I would like to thank Professor Fernando Kuipers for making this research possible, and Dr. Nitinder Mohan and Dr. Tanya Shreedhar for their daily supervision. Without their professional guidance, I would not have been able to complete this study.

I am also grateful to my classmates and friends, whose help allowed me to overcome many challenges.

A lot has happened in recent years. I want to thank my parents and Yuxin Sun. For many years, I have tried my best to repay their kindness and to fufill the responsibilities I ought to bear. However, in reality, they have silently shouldered much of my pain for me. I feel both guilty and deeply grateful for this.

On the edge of speech, yet held in silence. May the days to come be gentle and beautiful.

Zuji Zhou
Delft, May 2026

Abstract

This thesis presents Media over Multipath QUIC (MoMQ), a design that extends Media over QUIC Transport (MoQT) with multipath-aware object delivery. The core contribution of MoMQ is a rule-based mechanism that allows endpoints to install object-to-path mapping rules, enabling relays to schedule media objects across multiple network paths according to application-level delivery preferences. These rules operate on generic object metadata, allowing relays to remain application-agnostic while supporting fine-grained, semantics-aware media distribution.

The motivation for MoMQ stems from the limitations of single-path transport for emerging real-time media applications. High-resolution video, ultra-low-latency cloud gaming, and high-frame-rate video conferencing increasingly approach the performance limits of a single network path. Multipath QUIC provides a standards-compliant transport substrate that can aggregate heterogeneous network resources and improve resilience, making it a necessary building block for future real-time media systems. However, transport-layer-only multipath scheduling is insufficient to meet the strict latency and quality requirements of real-time media without guidance from application semantics.

MoMQ bridges this gap by exposing a controlled interface through which applications can express delivery preferences, while preserving MoQT's decoupled relay architecture. As a result, MoMQ can flexibly support diverse real-time applications, including live streaming and video conferencing, without binding relays to specific application logic.

To evaluate the proposed design, this thesis analyzes the stringent requirements of video conferencing under advanced encoding strategies such as Scalable Video Coding (SVC) and derives MoMQ scheduling policies accordingly. A prototype system is implemented and deployed in a real-world multipath environment consisting of a terrestrial WiFi link (representative of typical 4G LTE characteristics) and a Low Earth Orbit (LEO) satellite link. Transport-only baseline measurements confirm that existing multipath schedulers fail to improve upon single-path tail latency, motivating the need for application-level scheduling guidance. Four declarative MoMQ rules addressing P-frame interleaving, reconfiguration avoidance, dependency co-location, and cost-sensitive path preference collectively reduce P99.9 frame completion time by 71.4% compared to the best single-path baseline and by 70.3% compared to the best transport-only multipath scheduler, while routing only approximately 11.3% of traffic over the metered backup path.

Contents

Acronyms

BBR Bottleneck Bandwidth and Round-trip propagation time

BDP Bandwidth-Delay Product

CCA Congestion Control Algorithm

CWND Congestion Window

FCT Frame Completion Time

GEO Geostationary Earth Orbit

GOP Group of Pictures

GSL Ground-to-Satellite Link

ISL Inter-Satellite Link

IDR Instantaneous Decoder Refresh

LEO Low Earth Orbit

MoQT Media over QUIC Transport

MoMQ Media over Multipath QUIC

MPQUIC Multipath QUIC

QUIC Quick UDP Internet Connections

RTT Round-Trip Time

SVC Scalable Video Coding

1. Introduction

Real-time media traffic is growing fast, and the network has to keep up [10]. Applications like live streaming and video conferencing need high throughput and low latency while also coping with sudden path changes [1, 38]. As these applications aim for even lower latency, especially for interactive use [19], relying on just one path becomes more risky because there is almost no buffer left to handle network issues.

Multipath transport addresses this limitation. By spreading traffic across interfaces, a sender can aggregate bandwidth and reduce tail latency. Path redundancy also improves robustness, as the connection can continue over the remaining paths if one link fails. Multipath QUIC (MPQUIC) [12] brings multipath support into the QUIC ecosystem, with path management, per-path congestion control, and schedulers such as Round-Robin, MinRTT, and BLEST [14].

Transport-layer multipath protocols alone still have fundamental limitations. Traditional real-time protocols treat media as an opaque byte stream. The transport sees packets, but not what they contain. Media over QUIC Transport (MoQT) [30] breaks with this model. MoQT organizes media into named Objects forwarded through relays. Each Object can carry metadata, so the network sees something about the media it carries. A relay can use this metadata to make forwarding decisions based on what a piece of media *is*, not just its size. This relay model also scales better and allows caching at each hop, which matters for large live delivery.

Although MPQUIC and MoQT are complementary, a semantic gap remains. MPQUIC schedulers work on bytes and packets. They see RTT, loss, and congestion-window state, but cannot tell whether the bytes from MoQT belong to an IDR keyframe that anchors a whole Group of Pictures or to an enhancement-layer P-frame whose loss affects only one decoded image.

This semantic disconnect is especially consequential in video conferencing, one of the most latency-sensitive real-time applications. Many systems use Scalable Video Coding (SVC) [37], where a base layer carries the minimum decodable video and enhancement layers add resolution or frame rate. Different parts of the stream matter differently. Base layers are dependency roots. Enhancement layers are expendable. IDR frames are large and critical [43]. A scheduler that treats all bytes equally can make local decisions that still hurt the application. It might split dependent frames across paths with different delays, so the receiver waits for a base-layer frame on the slower path before it can decode. The problem is because the scheduler lacks application context.

A concrete example makes this gap visible. Consider a subscriber watching a live video conference over a laptop connected to both Starlink and a local WiFi hotspot. The conference stream carries SVC-encoded video with large IDR frames that reset the decoder, small P-frames that carry incremental updates, and enhancement layers that improve quality. Each frame type has different latency sensitivity, loss tolerance, and bandwidth requirements. The Starlink path offers high bandwidth but suffers periodic 50–80 ms disruptions during satellite reconfiguration, while the WiFi path has lower bandwidth but stable latency. An ideal scheduler would route IDR frames away from Starlink during reconfiguration windows, keep dependent frames together on the same path to avoid reordering, and default to the flat-rate satellite link to minimize metered WiFi cost. No existing transport-layer scheduler can make these decisions because the relevant information, such as frame types, dependencies, and cost preferences, is invisible at the transport layer.

This example shows why bridging this disconnect requires a careful separation of responsibilities. One option is to build application logic into the transport scheduler. That creates tight coupling. The scheduler would need updates for every new codec or format, and it could not serve different applications on the same relay. A more scalable alternative is to let the application express what matters, what can wait, and what should remain on the same path, while the transport layer satisfies those preferences under current conditions. That is the approach of this thesis.

This thesis combines multipath transport with application-level delivery semantics. Following the end-to-end argument [11, 35], application knowledge should inform network-layer decisions without replacing them. The result is MoMQ, a small extension to MoQT that lets endpoints express delivery preferences, such as priority or path preference, through object metadata. These preferences guide path selection at the relay. Relays stay codec-agnostic, they match rules against metadata fields and schedule accordingly, without needing to know what an IDR frame is or why a base layer matters.

1.1. Research Questions

Three research questions guide this thesis.

1. **How can application-level media semantics be communicated to a multipath transport relay without compromising relay scalability or application agnosticism?** MoQT relays are designed to be simple forwarding nodes that serve many subscribers and applications. If an extension requires the relay to understand media-format details such as SVC layer IDs or dependency graphs, that simplicity is lost and the relay becomes tied to those formats. The question is whether application-aware scheduling is possible while keeping the relay unaware of application semantics. Chapter 3 answers this with a match-action rule language where the relay evaluates rules mechanically against opaque metadata fields.
2. **What scheduling problems arise when SVC video conferencing traffic is sent over heterogeneous multipath networks without application guidance, and can declarative rules fix them?** SVC's layered encoding creates dependencies between video frames that transport-layer schedulers cannot see. An IDR frame that anchors an entire Group of Pictures gets the same treatment as a disposable enhancement-layer P-frame. When paths differ in RTT, capacity, or cost, this blindness leads to scheduling failures that hurt application quality. The goal is to identify which problems appear in practice and whether the rule framework from Question 1 can address them. Chapter 4 answers this through measurements on a real Starlink-WiFi testbed, identifying four distinct problems and deriving one rule for each.
3. **Can transport-agnostic delivery rules exploit LEO satellite path characteristics without requiring topology-specific configuration?** LEO satellite links introduce periodic beam handovers, latency that varies with orbital position, and cost structures that differ from terrestrial links. These disruptions follow predictable patterns. The question is whether a rule-based scheduler can turn that predictability into a scheduling advantage without hard-coding knowledge of a specific satellite constellation or its orbital parameters. Chapter 4 shows that path labels and reconfiguration-avoidance rules can express LEO-specific behavior as general scheduling preferences.

1.2. Contributions

This thesis makes three contributions.

First, it presents the MoMQ protocol [30], a small, additive extension to MoQT that adds multipath scheduling to the relay. Scheduling rules use object metadata and path labels rather than media formats or path identifiers, so the relay can evaluate them mechanically without understanding the media itself. The design adds only five wire-level elements to MoQT. Sessions that do not support MoMQ behave as standard MoQT, and the relay's own scheduling policies always take priority over the application's preferences. Chapter 3 presents the full design.

Second, the thesis provides a measurement-driven analysis of SVC video conferencing over a real Starlink LEO satellite link. It identifies four transport-level scheduling problems in SVC over heterogeneous multipath networks, traces them to a mismatch between what the transport sees and what the application needs, and derives MoMQ rules to address them. These problems appear consistently in real measurements, and existing multipath schedulers cannot fix them because they lack the application context that MoMQ provides. Chapter 4 covers the measurements, mechanisms, and evaluation.

Finally, the thesis integrates MoMQ with heterogeneous terrestrial and LEO satellite paths [28, 29]. LEO constellations bring latency variability, periodic beam handovers, and failure patterns that complement terrestrial links. Because these handovers follow a predictable schedule, MoMQ can steer traffic

away from a path before a disruption and return once conditions recover, without topology-specific configuration. Chapter 4 demonstrates this on a real Starlink-WiFi testbed.

1.3. Outline

Chapter 2 covers the technical background: MoQT's object model and relay architecture, MPQUIC's path management and scheduling, SVC's hierarchical encoding, and the network characteristics of LEO satellite constellations. These topics matter because MoMQ sits at their intersection.

Chapter 3 presents the MoMQ protocol design. It covers the four design principles, the architecture, session setup, rule syntax and semantics, the metadata model, the relay-side scheduling pipeline, and security considerations.

Chapter 4 applies MoMQ to SVC video conferencing on a real terrestrial-LEO multipath testbed. It presents the measurement study that reveals four transport-level scheduling problems, derives mechanisms to address them, specifies the corresponding MoMQ rules, and compares transport-only baselines against single-path references. The chapter also tests topology sensitivity: moving the subscriber to Canada shows that the gains nearly disappear when the subscriber–relay paths converge onto shared backbone routers, but moving the relay to Finland, where traceroute confirms distinct paths, reproduces the original results at a longer distance.

Chapter 5 summarizes the contributions, discusses limitations, and outlines future work on automated rule generation, broader application domains, and standardization.

2. Background and Related Work

This chapter provides the technical background required to understand MoMQ and its application to scalable video conferencing. The problem in this thesis brings together four main technologies: MoQT for media delivery, MPQUIC for sending data over multiple paths, SVC for video encoding, and LEO satellite networks as the environment. Each of these plays an important role, and their limitations help explain why MoMQ is designed the way it is.

The remainder of this chapter is organized into four sections corresponding to these topics. Each section concludes by relating the material to application-aware media transport. The presentation follows the system stack, beginning with the media protocol that structures the data, then the transport layer that carries it, the video encoding model that creates the scheduling challenge, and finally the network environment in which these challenges become most pronounced.

Therefore, Section 2.1 introduces Media over QUIC Transport, or MoQT, a new protocol for real-time media delivery. It explains how MoQT organizes data into objects and uses relays, which helps set the foundation for how MoMQ adds its rule mechanism. Section 2.2 looks at Multipath QUIC, which allows data to travel over multiple network paths. It focuses on how paths are managed, how data is scheduled, and how congestion is controlled. Section 2.3 explains Scalable Video Coding, a video encoding method where frames depend on each other in layers, showing why smarter scheduling based on application needs is important. Finally, Section 2.4 explores low Earth orbit satellite networks, highlighting challenges like changing latency, frequent reconfigurations, and the mix of satellite and terrestrial connections, which are used to test how well MoMQ performs in realistic conditions.

2.1. Media over QUIC Transport

Media over QUIC Transport (MoQT) is a publish-subscribe protocol for real-time media. It runs over QUIC [20] and WebTransport [30, 42, 5] and provides a unified object-based model across use cases that were previously split across separate protocol stacks, e.g., RTP-based conferencing [36] and HTTP-based adaptive streaming [38].

2.1.1. System Architecture

Figure 2.1 presents a four-layer stack where MoQT connects the application layer with the transport layer. Applications create objects along with their metadata, MoQT manages how these objects are announced, subscribed to, and forwarded, and QUIC handles secure and multiplexed data transport.

MoQT relies on relay-based distribution instead of sending data directly from publishers to subscribers, as shown in Figure 2.2.

Relays are central to this distribution model. They allow a single publisher connection to serve many subscribers, a role analogous to CDN edge servers [32]. They can cache recent objects to reduce join latency and retransmission overhead, enforce authorization at network boundaries, and make local forwarding decisions based on downstream conditions without publisher involvement. Despite these responsibilities, relays remain application-agnostic. They forward by protocol semantics rather than codec logic, and MoMQ preserves this property by matching generic metadata instead of relying on media-specific parsing.

2.1.2. Hierarchical Data Model

MoQT organizes media as **Track** → **Group** → **Subgroup** → **Object** (Table 2.1).

This hierarchy enables group-level random access, subgroup-level priority differentiation, and object-level caching for efficient retransmission. The stable addressing scheme (track, group, subgroup, object) also supports precise subscription and reference.

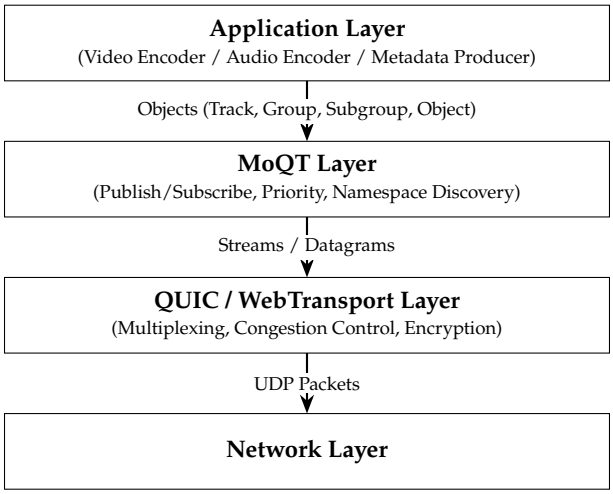

Figure 2.1: MoQT protocol stack architecture. The application layer produces media objects with semantic metadata. MoQT provides publish-subscribe semantics and priority-based delivery. QUIC handles reliable, encrypted, multiplexed transport.

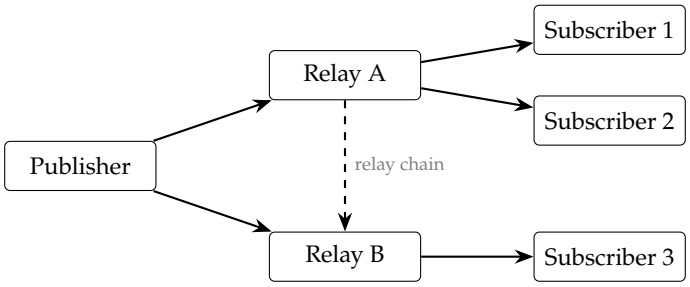

Figure 2.2: MoQT relay topology. The publisher distributes through multiple relays, each serving a subset of subscribers. Each subscriber connects to one relay. Relays may chain to each other (dashed), forming a relay graph that scales distribution, caches objects, and aggregates subscriptions.

2.1.3. Priority and Scheduling

MoQT defines two-level priority for ordering under congestion. Subscriber priority (0–255, lower is higher) expresses receiver preferences across tracks, while publisher priority per object or subgroup expresses inherent importance. The scheduler resolves ties deterministically: subscriber priority, then publisher priority, then group order to avoid staleness, and finally subgroup/object order to maintain production sequence. This priority model complements but differs from RTP's payload-type signaling [36], which lacks a relay-oriented scheduling framework.

This model decides *what* to send first, not *which path* to use. MoMQ adds this missing dimension via rule-driven multipath preferences.

2.1.4. Limitations for Multipath Transport

MoQT provides rich application-level semantics, namely named objects, hierarchical grouping, per-object priority and relay-based distribution, yet its transport model assumes a single path between each pair of adjacent nodes. This creates three practical limitations when the underlying network offers multiple paths.

First, all objects share the same path-level conditions. When a relay forwards both a high-priority IDR frame and a low-priority enhancement P-frame, both experience the same RTT, loss rate, and congestion state. If the path happens to be congested, the priority model can decide which object to send first, yet it cannot route the critical object to a less-congested alternative path.

Second, there is no path diversity for resilience. If the single path between a relay and its subscriber experiences a disruption, such as a routing change, a satellite reconfiguration, or a transient outage, all

Table 2.1: MoQT hierarchical data model with video codec mapping examples.

Level	Description	Video Example
Track	A named sequence of groups, identified by namespace and track name. Subscribers issue subscriptions at track granularity.	A video resolution layer (e.g., 1080p track, 720p track)
Group	A collection of objects that are independently decodable. Groups provide join points for late-joining subscribers.	A Group of Pictures (GOP) starting with a keyframe
Subgroup	Objects within a group sharing dependency and priority relationships. Subgroups map to QUIC streams for in-order reliable delivery.	A temporal layer (Layer 0, Layer 1, Layer 2)
Object	The atomic data unit with immutable payload. Each object is uniquely addressed by (namespace, track name, group ID, object ID).	An individual video frame

objects are affected equally. Multi-interface devices that have both WiFi and cellular connectivity, or both terrestrial and satellite links, cannot use the backup interface.

Third, subscribers are unaware of costs. A subscriber connected through both a flat-rate WiFi link and a metered LTE link cannot offload bulk traffic to the cheaper path or use the LTE link selectively for time-critical objects.

These limitations are not design flaws in MoQT, instead, they are a natural consequence of its single-path transport assumption. MoMQ bridges the gap by preserving MoQT's relay and priority semantics while adding rule-driven path selection underneath. Chapter 3 details this extension.

2.2. Multipath QUIC

Multipath QUIC (MPQUIC) allows one connection to use multiple paths in parallel [12]. It supports bandwidth aggregation, resilience to path failure, and smooth reconfiguration across interfaces. Compared with MPTCP [15], MPQUIC benefits from QUIC's encrypted signaling [41], migration support, and stream multiplexing, which together make multipath operation more deployable for media traffic.

2.2.1. Path Management

MPQUIC uses *path identifiers* and maintains independent state per path.

For each active path, MPQUIC maintains a path ID, the local/remote address tuple, RTT estimates (smoothed, minimum, and variance), congestion state (cwnd and bytes in flight), loss detection state, and a status flag (validated, active, standby, or abandoned). This design keeps heterogeneous links independent, maintaining separate RTT and congestion-window evolution for each link.

Path establishment uses "advertisement" and "validation". Endpoints advertise additional addresses via ADD_ADDRESS, initiate a new path by sending packets to an advertised address with validation tokens, complete validation via a `PATH_RESPONSE` handshake, and then activate the path for scheduling or standby use. Paths can be added and removed dynamically, enabling seamless mobility across WiFi, cellular, and satellite links.

2.2.2. Multipath Schedulers

The scheduler decides which path carries each packet. This directly affects throughput, latency, and reliability [34]. Four common strategies are summarized below.

Round-robin distributes packets cyclically across all available paths,

$$\text{path}(n) = n \mod |\mathcal{P}| \tag{2.1}$$

where n is the packet sequence number and $|\mathcal{P}|$ is the number of active paths.

Round-robin is simple yet ignores path asymmetry. With heterogeneous RTT, slow-path packets can trigger receiver head-of-line blocking. This is often acceptable for bulk transfer, not for real-time media.

The Minimum RTT (MinRTT) scheduler prefers the path with lowest observed round-trip time,

$$\text{path}^* = \arg\min_{p \in \mathcal{P}} \text{RTT}_p \tag{2.2}$$

MinRTT minimizes per-packet latency but can underuse slower-path capacity. It may oscillate quite often as the lowest RTT path alternates or starve a higher-bandwidth path if its RTT remains higher. Therefore, the performance of MinRTT depends on the degree of difference in the path and the characteristics of the upper-layer application. For example, large file downloads and RTC applications will result in different performance.

The Redundant scheduler transmits each packet on all available paths,

$$\forall p \in \mathcal{P} : \text{send}(\text{packet}, p) \tag{2.3}$$

This maximizes reliability and minimizes tail latency by using the first-arriving copy, but costs $|\mathcal{P}|\times$ bandwidth which is not always acceptable or necessary. For high-bitrate video, full duplication is usually too expensive, motivating selective redundancy for only critical content.

BLEST [14] estimates whether using a slower path will cause head-of-line blocking before deciding to send on it. More recent schedulers such as ECF [24] further refine heterogeneous-path scheduling by estimating per-path completion times.

BLEST's decision rule:

$$\text{send on slow path if: } \frac{\text{RTT}_{\text{slow}}}{2} < \frac{\text{cwnd}_{\text{fast}} - \text{inflight}_{\text{fast}}}{\text{rate}_{\text{fast}}} \tag{2.4}$$

BLEST balances latency and throughput better than simple heuristics and usually has better performance, yet it is still transport-only and cannot distinguish keyframes from delay-tolerant frames. Therefore, BLEST also has fundamental limitations.

2.2.3. Congestion Control Algorithms

Each path runs its own congestion control. For media, queueing delay is as important as throughput [22]. We summarize Cubic, BBR, and Copa. Other notable approaches include learning-based methods such as PCC Vivace [13].

Cubic [17] is a loss-based congestion control algorithm that uses a cubic function to grow the congestion window:

$$W(t) = C(t - K)^3 + W_{\max} \tag{2.5}$$

where $W_{\max}$ is the window size at the last congestion event, $K = \sqrt[3]{W_{\max}\beta/C}$ is the time to reach $W_{\max}$, C is a scaling constant, and β is the multiplicative decrease factor (typically 0.7).

Cubic is throughput-oriented but loss-driven, so it often builds queues before backing off. This inherently adds latency especially under contention, which is undesirable for interactive media.

Bottleneck Bandwidth and Round-trip propagation time (BBR) [7] takes a model-based approach, explicitly estimating the network's bandwidth-delay product (BDP) and operating at that optimal point:

$$\text{BDP} = \text{BtlBw} \times \text{RTprop} \tag{2.6}$$

where BtlBw (bottleneck bandwidth) is the maximum delivery rate observed and RTprop (round-trip propagation time) is the minimum RTT observed.

BBR alternates between probing bandwidth (sending around estimated BtlBw) and probing propagation delay (draining queues to measure true RTT). It targets the path BDP and usually keeps lower queues than Cubic, though probing and fairness interactions can still create short-term instability. In latency-sensitive scenarios, it is often preferable to Cubic.

Copa [2] is a delay-based algorithm explicitly designed to achieve both high throughput and low queuing delay. Copa optimizes a utility function combining throughput λ and delay d:

$$U = \log \lambda - \delta \log d \tag{2.7}$$

where δ controls the throughput-delay tradeoff. The parameter δ is typically set to 0.5, valuing delay reduction equally with throughput gains.

Copa derives the optimal sending rate by differentiating the utility function:

$$\lambda^* = \frac{1}{\delta \cdot d_q} \tag{2.8}$$

where $d_q = \text{RTT}_{\text{standing}} - \text{RTT}_{\text{min}}$ is the estimated queuing delay (current RTT minus the minimum observed RTT, which approximates propagation delay).

Copa adjusts its congestion window toward this target rate using a velocity parameter v that accelerates convergence:

$$\text{cwnd} \leftarrow \text{cwnd} \pm \frac{v}{\delta \cdot \text{cwnd}} \tag{2.9}$$

Copa is delay-oriented and explicitly balances throughput against queueing delay. Its competitive mode improves coexistence with loss-based flows. For multipath media, its lower queueing delay helps reduce inter-path skew and reordering.

2.2.4. Limitations for Media Transport

Despite the complexity of MPQUIC's scheduling and congestion control mechanisms, a fundamental limitation remains: the scheduler operates on bytes, not media objects. It knows how many bytes are queued on each path, what the current RTT and loss rate look like, and whether a path's congestion window has room for more data. What it does not know is what those bytes represent.

This blindness leads to several concrete problems. The scheduler cannot distinguish a base-layer frame whose loss prevents an entire GOP from decoding from an enhancement-layer frame whose loss causes only localized quality degradation. It cannot account for dependency order, e.g., if frame B depends on frame A, sending B on the low-latency path and A on the high-latency path forces the receiver to wait for A before it can use B, negating the latency benefit. And it cannot coordinate with MoQT relay logic because path-selection decisions happen below the relay layer, invisible to caching, aggregation, and downstream adaptation.

The result is that MPQUIC schedulers make locally rational decisions. For example, MinRTT chooses the fastest path, BLEST avoids paths that would cause blocking, Round-Robin spreads load evenly, but these decisions can be globally suboptimal for media delivery. Chapter 4 demonstrates this concretely, none of the four standard schedulers improves upon single-path tail latency for SVC video. The missing piece is application-level guidance, a way for the media layer to tell the transport which objects matter most and how they should be distributed across paths. MoMQ provides exactly this interface.

Scalable Video Coding provides a concrete example of this mismatch because frame dependencies and frame-size asymmetry translate directly into scheduling consequences.

2.3. Scalable Video Coding

Scalable Video Coding (SVC) encodes video into hierarchically dependent layers, enabling adaptation to network and device constraints [37]. SVC extends the H.264/AVC standard [43], with analogous

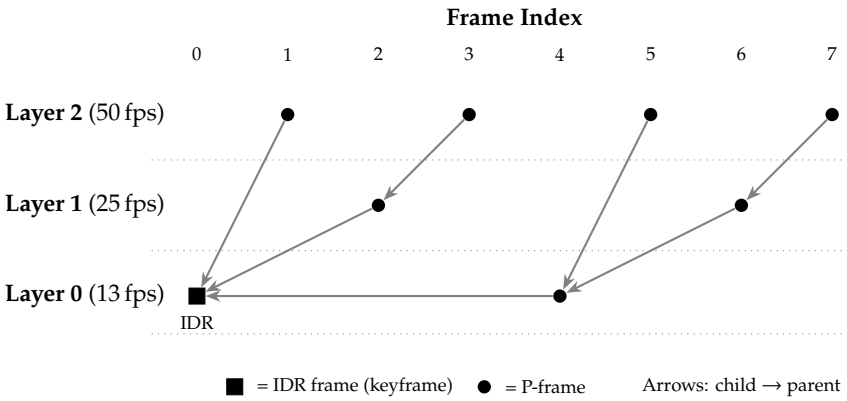

Figure 2.3: Three-layer temporal scalability structure at 50 fps base encoding. A 50-frame GOP contains 13 Layer 0 frames (1 IDR + 12 P, every 4th position), 12 Layer 1 frames, and 25 Layer 2 frames, yielding cumulative rates of 13, 25, and 50 fps. Each P-frame depends on an earlier frame in the same or lower layer. The GOP structure repeats, with each GOP starting from an IDR frame.

scalability extensions defined for HEVC [39]. Instead of switching among independent encodings, SVC can adapt at delivery time by selecting which layers to forward.

2.3.1. Scalability Dimensions

SVC supports temporal, spatial, and quality (SNR) scalability. Temporal scalability varies frame rate by layering frames, dropping higher layers reduces frame rate without stalling playback. Spatial scalability provides multiple resolutions, dropping higher spatial layers reduces sharpness while preserving content. SNR scalability refines quality at fixed resolution, dropping those layers introduces artifacts but maintains resolution and frame rate. These dimensions can be combined. This thesis focuses on temporal scalability because it provides clear dependency and priority structure for multipath scheduling.

2.3.2. Temporal Layer Structure

Figure 2.3 illustrates a three-layer temporal scalability structure within a Group of Pictures (GOP). Each GOP begins with an IDR (Instantaneous Decoder Refresh) frame that provides a random access point, it is a frame that can be decoded without reference to any previous frame. Subsequent P-frames (predicted frames) are organized into temporal layers with hierarchical dependencies.

Decodability follows strict dependency rules. Layer 0 contains IDR frames and key P-frames at the lowest frame rate, IDR frames have no dependencies, and Layer 0 P-frames depend only on earlier Layer 0 frames. Layer 1 P-frames depend on the nearest preceding Layer 0 frame, and Layer 2 P-frames depend on the nearest preceding Layer 1 or Layer 0 frame. Dropping higher layers reduces frame rate while preserving lower-layer playback. Without Layer 0, no frames decode.

Lower layers are prerequisites for higher layers, directly affecting transport transmission. Delivering a Layer 2 frame before its Layer 0 dependency wastes bandwidth, base-layer loss cascades to dependent frames, and IDR loss can invalidate an entire GOP (often 1–2 seconds of video).

2.3.3. Frame Size Characteristics

A key SVC property is frame-size asymmetry: IDR frames are much larger than P-frames [43].

Table 2.2 presents representative frame sizes for 1080p SVC encoding at 50 fps measured from the evaluation setup described in Chapter 4. The IDR frame is approximately 23× larger than a typical P-frame. This size difference has several transport-level consequences that affect scheduling decisions.

First, IDR frames create bursty traffic as they are fairly big. A single IDR frame of 230 KB requires about 161 packets at a typical QUIC maximum datagram size of 1430 bytes. At 100 Mbps, transmitting these packets already takes approximately 18 ms, which is a non-trivial fraction of the 150 ms interactive latency budget. If the congestion window is smaller than the IDR frame (which is typical for delay-sensitive

Table 2.2: Typical frame sizes for 1080p SVC encoding at 50 fps with QP-based quality differentiation (QP 18/26/42 for Layer 0/1/2).

Frame Type	Size	Packets (1430B)	Frequency
IDR frame	~230 KB	161 packets	1 per GOP (~1/sec)
Layer 0 P-frame	~15 KB	11 packets	12 per GOP
Layer 1 P-frame	~10 KB	7 packets	12 per GOP
Layer 2 P-frame	~8 KB	6 packets	25 per GOP

congestion control algorithms like Copa), the frame must be transmitted over multiple congestion-window rounds, each adding one RTT of delay. For a 3-round transmission over a 45 ms path, the IDR frame alone takes 135 ms to complete, leaving almost no time for downstream processing.

Second, IDR frames face much higher risks of loss. At a 3% packet loss rate, a 161-packet IDR frame expects approximately 5 lost packets, requiring retransmission and adding at least one additional RTT per lost packet. A 7-packet P-frame, by contrast, expects only 0.2 lost packets on average. The combination of more retransmissions and a larger base size means that IDR frame completion times can easily exceed 100 ms, making them a critical bottleneck.

Third, IDR frame loss has cascading consequences. Because all subsequent frames in the GOP depend on the IDR frame directly or indirectly through the dependency chain, losing an IDR frame effectively invalidates the entire 1-second GOP which is up to 50 frames of video. Losing a Layer 2 P-frame, by contrast, affects only that single frame at the highest temporal resolution. This asymmetry in loss impact is invisible to a transport scheduler that treats all packets equally.

These characteristics make uniform byte-level scheduling inherently inefficient for SVC. Treating IDR packets the same as enhancement P-frame packets will inherently ignore a 23:1 difference in criticality and a 50:1 difference in loss impact.

2.3.4. Network Adaptation via Layer Selection

Temporal scalability enables frame-rate adaptation by dropping higher layers under constrained bandwidth as shown in Table 2.3. This approach is similar to bitrate adaptation in HTTP adaptive streaming systems [38, 3] but does not require a separate manifest or segment-based delivery.

This degradation is graceful. Dropping enhancement layers reduces smoothness first, while preserving base continuity. For multipath transport, this motivates stronger protection for lower layers than for higher layers.

2.3.5. Implications for Multipath Transport

SVC's dependency structure creates transport requirements that go beyond what byte-level schedulers can provide [34]. Several properties of SVC encoding are particularly relevant for multipath scheduling.

Loss tolerance asymmetry. Loss in the base layer is critical. If a Layer 0 frame is missing, any higher-layer frame that depends on it cannot be decoded. In contrast, losing enhancement layers is less severe, as it only reduces video quality in certain parts instead of breaking the whole stream. Yet transport schedulers see only bytes and cannot apply selective reliability. They cannot choose to retransmit base-layer packets more aggressively or to duplicate IDR frames across multiple paths while letting enhancement frames travel on a single path.

Latency asymmetry. Base layers have stricter latency constraints than enhancement layers. Delaying a

Table 2.3: Adaptive frame rate selection based on network conditions.

Network Condition	Layers Received	Frame Rate	Bitrate
Excellent	0, 1, 2	50 fps	100%
Moderate	0, 1	25 fps	~73%
Poor	0 only	13 fps	~56%

Layer 0 frame stalls all higher-layer frames within the same GOP that depend on it either directly or indirectly, amplifying a single-frame delay into a multi-frame stall. Delaying a Layer 2 frame affects only itself. This creates a risk in multipath scheduling, if enhancement packets happen to traverse a fast path while base packets are stuck on a slow path, the receiver cannot make use of the early-arriving enhancement frames until the base frames catch up.

IDR vulnerability. The bursty nature of IDR frames makes them particularly vulnerable to transient disruptions. A brief outage of 50–100 ms, which is well within the duration of a satellite reconfiguration, can corrupt or delay an IDR frame, invalidating an entire GOP. Transport-layer multipath scheduling is typically reactive, it responds to measured loss and latency rather than anticipating upcoming disruptions. SVC conferencing benefits from proactive scheduling that shifts IDR traffic away from a path *before* a predicted disruption occurs rather than recovering afterward.

Dependency-order constraints. When dependent frames are split across paths with different RTTs, the receiver must buffer early-arriving frames until their prerequisites arrive on the slower path. This reordering penalty is predictable from the dependency graph but invisible to the transport layer, which does not know that two frames are causally related.

Application-aware scheduling addresses these constraints by exposing layer and frame semantics to the path-selection process. It enables policies that route base layers on low-latency paths, avoiding sending IDR frames to the Starlink path during predicted disruptions, co-locate dependent frames on the same path, and degrade higher layers first under congestion. Chapter 4 applies these policies in a complete SVC conferencing strategy using MoMQ rules.

2.4. LEO Satellite Networks

LEO constellations (500–2000 km) provide global broadband with much lower latency than GEO systems [18]. This section summarizes LEO properties relevant to multipath media transport.

Relative to GEO (typically 500–700 ms RTT), LEO enables interactive use but introduces several new problems such as frequent reconfigurations, variable geometry, and dynamic link quality.

2.4.1. System Architecture

Figure 2.4 illustrates the LEO satellite network architecture. The system includes three major components: user terminals, satellites, and ground infrastructure.

User Terminals are ground-based antennas that track and communicate with overhead satellites. Modern terminals (e.g., Starlink Dishy) use electronically steered phased-array antennas that can rapidly switch between satellites without mechanical movement. Terminals handle frequency translation, modulation, and increasingly sophisticated link adaptation to maintain connectivity as satellites pass overhead.

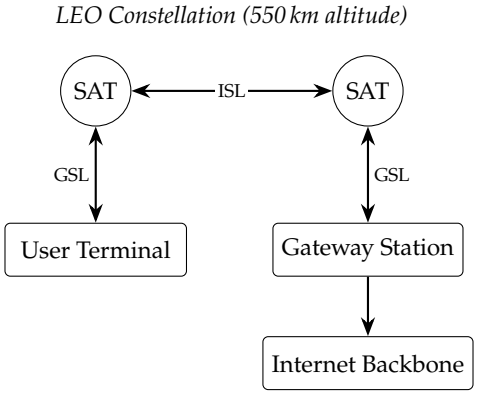

Figure 2.4: LEO satellite network architecture. User terminals communicate with overhead satellites via ground-to-satellite links (GSL). Satellites relay traffic to gateway stations via GSL, or increasingly via inter-satellite links (ISL) to reach distant gateways. Gateways connect to the terrestrial Internet backbone.

Table 2.4: Comparison of network path latencies across access technologies.

Path Type	Typical RTT	Variability	Notes
Terrestrial Fiber	10–30 ms	Low	Dominated by distance, minimal queuing
Cellular (4G/5G)	20–50 ms	Medium	Varies with load, reconfiguration events
LEO Satellite	25–80 ms	High	Varies with satellite position, ISL hops
GEO Satellite	500–700 ms	Low	Fixed distance dominates

Satellites receive uplink traffic from user terminals and forward it toward the destination. In first-generation LEO systems, each satellite serves as a simple "bent pipe," relaying traffic to a ground station within its footprint. Second-generation satellites incorporate inter-satellite links (ISLs) to enable traffic to traverse the constellation without touching ground until reaching a gateway near its destination [4]. ISLs are critical for serving users far from gateway stations (e.g., from Germany to South Africa) and for reducing reliance on dense ground infrastructure.

Gateway Stations are high-capacity ground facilities that connect the satellite constellation to the terrestrial Internet. Each gateway serves multiple satellites, aggregating traffic from potentially thousands of user terminals. Gateway placement is strategic. Dense placement in population centers reduces the average hop count to Internet peering points, while placement in remote areas (enabled by ISLs) extends coverage globally.

Unlike GEO, LEO satellites move relative to users, so continuous service requires frequent reconfigurations between satellites.

2.4.2. Network Characteristics

LEO links differ from terrestrial links in ways that directly affect multipath scheduling.

LEO satellites provide substantially lower latency than geostationary satellites, but higher and more variable latency than most terrestrial links as shown in Table 2.4.

LEO RTT varies with satellite elevation, the number of inter-satellite hops, gateway distance, and congestion. Overhead satellites reduce path length, multi-hop ISLs add delay, remote gateways increase terrestrial RTT, and queueing under load can add tens of milliseconds.

Operational measurements report about 25–80+ ms RTT depending on geometry, gateway path, and load [28, 29]. This variability complicates latency-sensitive media.

LEO systems can deliver high throughput (often 100–300 Mbps peak), but bandwidth varies with beam congestion, elevation angle, weather (rain fade in Ka-band), and per-satellite capacity limits. These effects can drop per-user rates to 10–50 Mbps at peak load and introduce short outages.

This variability makes rate control and adaptation harder than on fairly stable terrestrial links.

LEO also shows periodic non-congestive loss patterns [21, 40]. Reconfiguration gaps and intra-satellite beam switching cause brief disruptions. Weather produces sudden loss spikes, and line-of-sight obstructions create outages as satellites move across the sky. These losses are not congestion-related, so naive loss-based backoff can be ineffective [23]. Delay-based methods might help, but high RTT variability remains challenging.

2.4.3. Multipath Considerations

For dual-connected devices (LEO + terrestrial), multipath offers clear benefits and non-trivial scheduling trade-offs as shown in Table 2.5.

These differences create opportunities that naive aggregation cannot exploit well. Consider a subscriber connected to both a Starlink terminal and a WiFi access point. The WiFi path typically provides lower and more consistent latency, making it the natural choice for latency-critical base-layer frames. The satellite

Table 2.5: Complementary characteristics of satellite and terrestrial paths that motivate intelligent multipath scheduling.

Characteristic	LEO Satellite	Terrestrial (Cellular)
Coverage	Global (with constellation)	Limited to infrastructure
Latency	25–80 ms, variable	20–50 ms, moderate variability
Bandwidth	50–200 Mbps, variable	10–100 Mbps, variable
Reliability	Periodic reconf. gaps	Congestion-induced loss
Cost	Often flat-rate	Frequently metered

path offers high throughput at flat-rate pricing, making it well-suited for bulk enhancement-layer data. A transport-layer scheduler that assigns packets by RTT alone (MinRTT) will route almost everything to WiFi, leaving satellite capacity unused. One that tries to distribute packets evenly like Round-Robin will cause cross-path reordering because WiFi packets arrive 20–30 ms earlier than those sent on the satellite path. Neither approach accounts for the asymmetry in a way that benefits the application.

Reconfiguration gaps add another dimension the transport layer cannot handle alone. When a satellite beam reconfiguration disrupts the LEO path for 50–170 ms, any packets in flight on that path will be delayed or lost. If those packets belong to an IDR frame, the delay affects an entire GOP. Cost asymmetry presents a third challenge. If the cellular backup is metered per gigabyte while the satellite link is flat-rate, bulk traffic should remain on the satellite to minimize cost. The transport layer has no concept of such cost, it optimizes for throughput and latency, not billing. Only the application layer can express the preference that enhancement-layer frames should use the cheap path while base-layer frames may use the expensive path when latency demands it.

2.4.4. Relevance to This Thesis

LEO networks are a strong testbed for application-aware scheduling because they combine heterogeneity and dynamics [29]. Satellite and terrestrial paths differ in RTT, bandwidth, and loss behavior. A 4× RTT disparity is common. Periodic reconfiguration and other non-congestive loss patterns also differ fundamentally from congestion loss. Operational constellations therefore enable real-world validation that laboratory emulation cannot match.

Chapter 4 evaluates MoMQ on a Starlink + WiFi multipath setup under these conditions, complementing earlier 5G network characterizations [31] with satellite-terrestrial measurements.

2.5. Related Approaches to Application-Transport Integration

Prior work has approached application-transport integration from several directions. This section highlights only the distinctions most relevant to MoMQ, while Table 2.6 summarizes the comparison.

2.5.1. Application-Level Framing and End-to-End Arguments

MoMQ follows two classic principles. Application Level Framing (ALF) [11] argues that protocols should operate on application data units rather than opaque byte streams, and the end-to-end argument [35] argues against embedding application-specific logic in the network. MoQT's Objects provide the ADUs, while MoMQ keeps policy at the endpoint and limits the relay to mechanical rule evaluation over metadata.

2.5.2. Cross-Layer Optimization for Video

GCC [9] and Salsify [16] show that media-aware transport can improve latency, but both operate in single-path settings and do not address relay-side path selection. MVFST [27] adds application-aware stream ordering within QUIC, yet still only addresses the "what to send first" dimension. MoMQ extends this line of work to multipath relay scheduling.

2.5.3. Quality of Service and Traffic Classification

DiffServ [6] and HTTP/3 priorities [33] provide advisory prioritization, but at coarser granularity than MoMQ. They can express urgency, not object-level distinctions such as IDR versus enhancement frames or path co-location of dependencies.

2.5.4. SDN-Inspired Media Scheduling

MoMQ borrows the match-action structure of OpenFlow [26], but applies it to Objects rather than flows. The relay evaluates application metadata instead of Layer 2–4 headers, enabling per-frame scheduling decisions without interpreting codec semantics.

2.5.5. Summary

Table 2.6 summarizes the comparison.

Table 2.6: Comparison of MoMQ with related approaches to application-transport integration.

Approach	**Multipath**	**Per-Object**	**App-Agnostic Relay**	**Path Selection**
Salsify	✗	✓	✗	✗
GCC (WebRTC)	✗	✗	✗	✗
DiffServ	✗	✗	✓	✗
SDN / OpenFlow	✗	✗	✓	✓
HTTP/3 Priorities	✗	✓	✓	✗
MPQUIC Schedulers	✓	✗	✓	✓
MoMQ	✓	✓	✓	✓

MoMQ is the only approach in this comparison that combines multipath transport, per-object scheduling, application-agnostic relay operation, and explicit path selection.

3. MoMQ Protocol Design

This chapter presents the design of Media over Multipath QUIC (MoMQ), a protocol extension that enables application-guided scheduling of MoQT Objects across multiple network paths. The design addresses the central challenge identified in Chapter 2 that transport-layer multipath schedulers cannot exploit application-level semantics, while application-layer protocols like MoQT cannot influence path selection. The motivating example in Chapter 1 already makes this challenge concrete. This chapter turns that intuition into a protocol design.

MoMQ introduces a signaling channel between the subscribing endpoint and its last-hop relay, consisting of one setup parameter (`ENABLE_MOMQ`) and four control messages. The relay reports available paths and their properties to the subscriber via `PATH_STATE_REPORT`. The subscriber can label paths with local knowledge, such as cost class or preferred usage, via `PATH_LABEL_UPDATE`, and installs declarative scheduling rules via `PATH_MAPPING_RULE` that match Objects by metadata and express delivery preferences. The relay acknowledges each rule with `PATH_MAPPING_RESULT`, evaluates installed rules against each outgoing Object, and attaches the resulting scheduling directive to the QUIC stream so that the Multipath QUIC scheduler can factor application intent into path selection. MoMQ does not modify MoQT or MPQUIC, and it operates entirely within the last-hop relay-to-subscriber segment of a single MoQT session.

The remainder of this chapter is organized as follows. Section 3.1 establishes four design principles that guide the specification. Section 3.2 provides a high-level architectural overview, covering the last-hop architecture, session lifecycle, path-label mechanism, and subscriber influence model. Section 3.3 specifies session establishment: the `ENABLE_MOMQ` setup parameter for capability negotiation, `PATH_STATE_REPORT` for path discovery, and `PATH_LABEL_UPDATE` for subscriber-side path annotation. Section 3.4 defines the `PATH_MAPPING_RULE` and `PATH_MAPPING_RESULT` messages for expressing and acknowledging delivery preferences. Section 3.5 describes the Object metadata model that rules operate upon. Section 3.6 specifies the three-stage cross-layer scheduling pipeline at the relay. Section 3.7 addresses security considerations. Finally, Section 3.8 summarizes the protocol elements.

3.1. Design Principles

The design of MoMQ is guided by four core principles that address the limitations identified in Chapter 2 while preserving MoQT's architectural strengths. These principles constrain the solution space and motivate the design choices in subsequent sections.

3.1.1. Application-Agnostic Relay Architecture

MoQT's relay-based architecture enables scalable content delivery by allowing intermediate nodes to cache and forward Objects without understanding application semantics [32]. A relay serving live sports streaming operates identically to one serving video conferencing, as it matches subscriptions, caches objects, and forwards based on priorities, without parsing codecs or understanding frame dependencies. This application-agnostic property is central to MoQT's value proposition that deploy relays once and serve diverse applications.

MoMQ preserves this property. Relays do not interpret the *meaning* of metadata values or understand the application's media format. A relay sees a rule matching objects where `frame_type` equals `IDR`; it does not know what an IDR frame is or why it matters. The relay's responsibility is mechanical, it just matches the condition and executes the action.

This separation of concerns improves deployment simplicity, security isolation, and performance predictability. Relay operators need not update software when new applications emerge because each application can define rules against the same infrastructure. Security benefits from the fact that relays do not execute application code or interpret complex formats, reducing the attack surface. Performance

remains predictable because rule matching is a bounded operation built on string comparisons and numeric range checks.

The application-agnostic principle does not prevent specialized behavior, it requires that specialization be expressed through generic rule mechanisms rather than hardcoded logic. A relay that "understands" SVC is one configured with rules that match SVC frame types, and the relay itself remains format-agnostic.

3.1.2. Transport-Agnostic Rule Expression

To reduce potential conflicts with the transport, a critical design decision is that mapping rules must not reference explicit path identifiers or network topology information. Rules express *what* the application wants (e.g., high priority, co-location, path preference) rather than *how* to achieve it (e.g., send on path 0, duplicate on paths 1 and 2). Path labels provide a layer of indirection: rules reference labels such as `cost_class=free` rather than path IDs, ensuring portability across network configurations. This follows the end-to-end design principle [35] and the Application Level Framing approach [11], which advocate placing application-specific knowledge at endpoints rather than in the network.

This constraint serves multiple purposes. It ensures portability across network configurations, for instance, if a subscriber moves from a WiFi + cellular setup to a WiFi + satellite setup, the same rules apply because they reference path labels ("cost_class", "link_type") rather than path IDs that would change with the network topology. It protects privacy by avoiding topology disclosure as the subscriber does not need to know the relay's internal path numbering or routing tables. And it preserves relay autonomy in translating preferences to path decisions, so the relay can adapt to conditions the subscriber cannot observe.

This design also holds under dynamic conditions. When a path degrades or fails, transport-agnostic rules remain valid, the relay simply re-evaluates which paths satisfy the label-based preferences given current measurements. There is no need for the subscriber to re-issue rules or for the protocol to signal path-level changes, because the rules never referenced specific paths in the first place.

Rules express delivery *preferences* rather than explicit path assignments. The preference types include priority (scheduling urgency), balancing (single-path versus multipath distribution), path preference (bias toward paths with a matching label), and path affinity (co-location with a related Object). The relay translates these preferences to path decisions based on live path state and local policy, so the application's intent is preserved while the implementation adapts.

3.1.3. Minimal Protocol Extension

MoMQ introduces one setup parameter and four control messages. The `ENABLE_MOMQ` setup parameter declares multipath-aware scheduling capability during session establishment [20]. `PATH_STATE_REPORT` and `PATH_LABEL_UPDATE` exchange path state and label information between relay and subscriber. `PATH_MAPPING_RULE` installs or removes object-to-path mapping rules, and `PATH_MAPPING_RESULT` acknowledges those operations with success or error status.

This minimalism is intentional. Protocol extensions impose costs on all implementations, even those that do not use the new features, including specification complexity, implementation effort, testing burden, and potential for new bugs. Each additional message type must be parsed, validated, and handled, even if only to reject it.

Minimalism also facilitates incremental deployment. A client implementing only `ENABLE_MOMQ` negotiation can operate with relays that support full MoMQ, with graceful fallback to transport-layer multipath. The extension is strictly additive, sessions that do not negotiate MoMQ capability operate identically to standard MoQT.

The four messages form two natural pairs, state exchange (`PATH_STATE_REPORT` and `PATH_LABEL_UPDATE`) and rule management (`PATH_MAPPING_RULE` and `PATH_MAPPING_RESULT`). Rules can be installed and removed, and operations are acknowledged with success or specific error codes. No separate query or enumeration messages are needed, reducing state synchronization complexity.

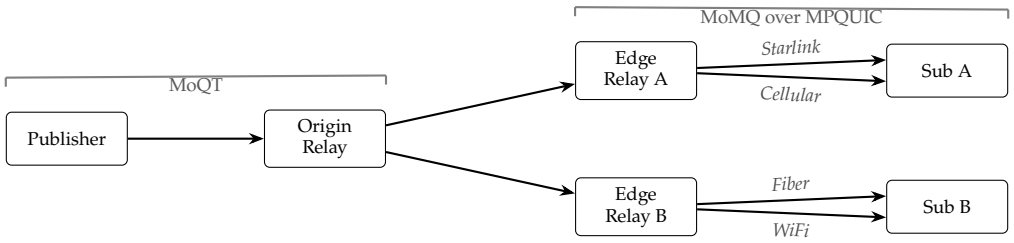

Figure 3.1: End-to-end MoQT delivery with MoMQ. Upstream hops carry standard MoQT sessions. MoMQ operates at the last hop where the relay–subscriber connection uses Multipath QUIC with disjoint network paths.

3.1.4. Advisory Semantics

Mapping rules are advisory hints rather than strict requirements. Relays implement rules according to local policy and resource availability, and applications cannot demand specific scheduling outcomes. This approach mirrors the Differentiated Services architecture [6], where per-hop behaviors are advisory and enforcement is domain-local.

This advisory nature stems from the fact that relays have real-time visibility into path conditions, while applications do not: a "lowest-latency" rule is best served by live RTT measurements. Resource constraints might also create conflicts (e.g., two flows requesting highest priority when only one path is healthy), and relay operators may cap redundancy or reserve paths for specific traffic classes. Advisory semantics allow such policies without protocol violation and accommodate varying relay capabilities.

3.2. Protocol Overview

This section summarizes MoMQ's architecture and operation before the detailed specification in subsequent sections.

3.2.1. Last-Hop Architecture

MoQT delivers media through a relay graph. A publisher injects Objects into an origin relay, which forwards them through intermediate relays to edge relays serving subscribers. Each hop is an independent QUIC connection carrying a MoQT session. At the last hop, the edge relay and subscriber may use a Multipath QUIC connection whose paths traverse disjoint networks, for example, one path over Starlink and another over cellular. MoMQ operates exclusively at this segment (Figure 3.1).

The subscriber installs rules on its edge relay to advise how Objects are distributed across MPQUIC paths. Each subscriber–relay pair runs an independent MoMQ session while upstream relays and the publisher are unaware of MoMQ. This scoping decision reflects the observation that multipath path diversity is most valuable at the network edge, where the subscriber has direct knowledge of local interface properties. Upstream hops between origin and edge relays typically traverse datacenter or backbone networks where path diversity is managed at the infrastructure level and where individual subscribers have neither visibility nor authority to influence routing.

The last-hop scope also simplifies deployment as only the edge relay needs to implement MoMQ, while origin relays and publishers continue to operate unmodified. A single relay can serve both MoMQ-enabled subscribers and legacy subscribers at the same time, since MoMQ affects only the last-hop segment.

The architecture is most effective when paths are largely disjoint (e.g., satellite vs. cellular), since the subscriber can direct latency-sensitive Objects to the lower-delay path while keeping bulk data on the higher-capacity one. Even when paths share some infrastructure, the subscriber's local knowledge of interface characteristics provides information that the relay cannot obtain from transport-level measurements alone.

3.2.2. Session Lifecycle

A MoMQ-enabled session proceeds through six phases (Figure 3.2).

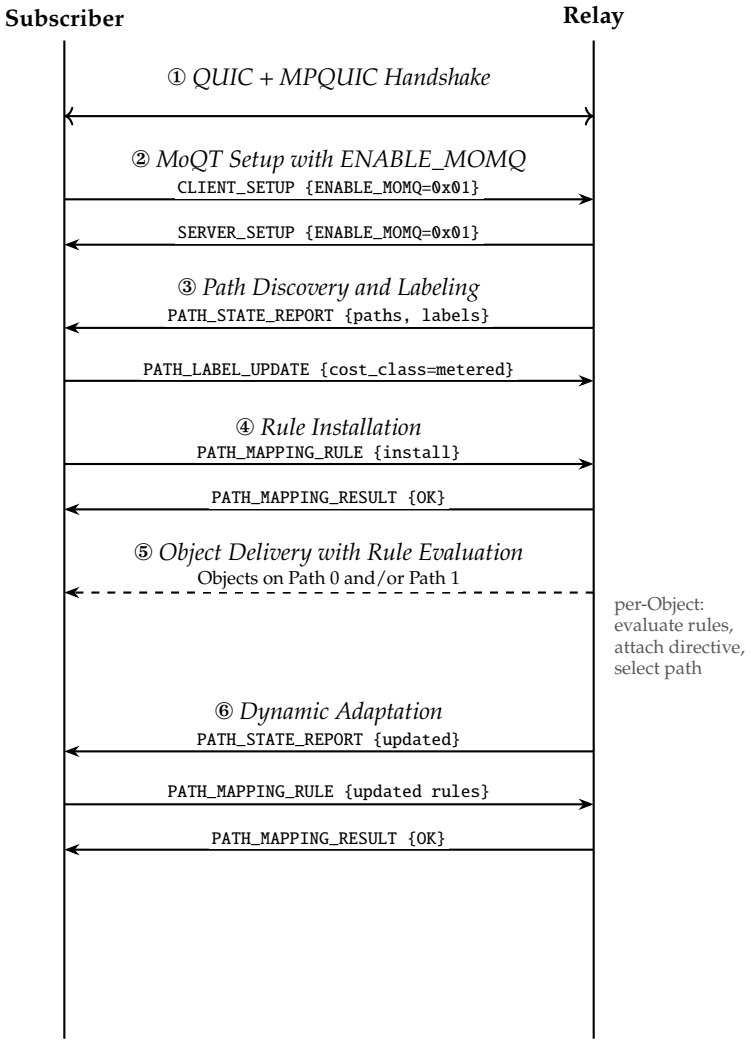

Figure 3.2: MoMQ session lifecycle. After QUIC and MoQT setup, the relay reports path state, the subscriber labels paths and installs rules, and Objects are delivered with per-Object rule evaluation. Path state changes trigger dynamic adaptation.

Phase ① establishes the QUIC connection with Multipath QUIC negotiation. Phase ② exchanges MoQT setup messages, both endpoints include `ENABLE_MOMQ` to activate MoMQ. Phase ③ begins path discovery, the relay sends a `PATH_STATE_REPORT` listing available paths and their relay-assigned labels, and the subscriber optionally sends `PATH_LABEL_UPDATE` messages to annotate paths with local knowledge (e.g., marking a path as metered). Phase ④ installs scheduling rules via `PATH_MAPPING_RULE`/`PATH_MAPPING_RESULT` exchanges. Phase ⑤ is steady-state delivery that for each outgoing Object, the relay evaluates rules against the Object's metadata, produces a scheduling directive, and hands it to the MPQUIC scheduler for path selection. Phase ⑥ handles dynamic adaptation, when path state changes (paths added, removed, or degraded), the relay sends an updated `PATH_STATE_REPORT`, and the subscriber may adjust rules accordingly. Phases ⑤ and ⑥ continue concurrently for the session's lifetime.

3.2.3. Path Labels

Rules never reference path identifiers directly to reduce conflicts as mentioned above. Instead, they reference *labels*: key-value pairs that describe path properties such as link type, cost class, geographic region, or any application-defined attribute.

Labels come from two sources. The relay assigns labels from operator configuration or transport observation (e.g., `link_type=satellite`) and reports them via `PATH_STATE_REPORT`. The subscriber assigns labels based on local knowledge (e.g., `cost_class=metered`) via `PATH_LABEL_UPDATE`. The relay merges both sets per path. When both sides assign the same key, the subscriber's value takes precedence.

This indirection makes rules portable. A rule that says "prefer paths where `cost_class=free`" works regardless of which physical path is free at any given moment. When the subscriber relabels paths in response to changing conditions, existing rules apply to the new topology without modification.

The label mechanism also supports dynamic reconfiguration scenarios. If a subscriber initially labels the Starlink path as `cost_class=free` but later switches to a metered satellite plan, it sends a single `PATH_LABEL_UPDATE` changing the label to `cost_class=metered`. All rules that reference cost class immediately adjust their behavior without any rule reinstallation. Similarly, a relay might label a path as `link_type=satellite` based on operator configuration, and the subscriber might add `stability=variable` based on observed reconfiguration frequency. The combined label set gives rules access to both operator knowledge (link type) and subscriber knowledge (observed stability), enabling scheduling decisions that neither side could make alone.

3.2.4. Subscriber Influence Model

MoMQ gives the subscriber two control planes operating through the same signaling channel. `PATH_LABEL_UPDATE` controls per-path behavior, the subscriber assigns labels that shape the environment in which scheduling operates. For example, a label that changes a path's congestion control algorithm alters its throughput and delay characteristics. `PATH_MAPPING_RULE` controls per-Object scheduling, rules express which Objects are urgent, which paths are preferred, and which Objects should travel together. The two planes are complementary, labels shape the environment, and rules express intent within that environment.

MoMQ state is session-scoped, negotiated capability, installed rules, and subscriber-assigned labels persist only for the session lifetime. Rules are discarded at session termination, are not persisted across sessions, and are isolated between subscribers. MoMQ composes with existing MoQT features. Subscriber priority remains inter-subscriber ordering, while MoMQ priority applies within a subscriber's own Objects. Group ordering, object status, caching, and stream-mapping modes are unchanged. MoMQ adds only path-selection guidance at forwarding time.

3.3. Session Establishment

Before rule-guided scheduling can operate, endpoints negotiate MoMQ capability during MoQT setup, and the relay reports its path topology to the subscriber. This section specifies the three messages that establish a MoMQ session, namely `ENABLE_MOMQ` for capability negotiation, `PATH_STATE_REPORT` for path discovery, and `PATH_LABEL_UPDATE` for subscriber-side path annotation.

3.3.1. The ENABLE_MOMQ Setup Parameter

MoMQ uses one setup parameter, `ENABLE_MOMQ` (key `0x10`, value `0x01`), carried in `CLIENT_SETUP` and `SERVER_SETUP` [30]. MoMQ is active if and only if both endpoints include this parameter. When active, the relay MUST send an initial `PATH_STATE_REPORT`, the subscriber MAY send `PATH_LABEL_UPDATE` and `PATH_MAPPING_RULE` messages. An endpoint that receives a MoMQ control message without prior negotiation MUST close the session with a Protocol Violation error.

MoMQ and Multipath QUIC are negotiated independently. A session may negotiate MoMQ even with a single MPQUIC path, for example, the `PRIORITY` action still governs scheduling order, and rules apply automatically if paths are added later. This decoupling is intentional because it allows applications to install rules before multipath is established, so that when a second path becomes available, scheduling preferences are already in place and take effect immediately without an additional round trip for rule installation.

If one endpoint includes `ENABLE_MOMQ` in the format shown in Figure 3.3 and the other does not, MoMQ is not activated, and the session proceeds as a standard MoQT session. The requesting endpoint receives no error, MoMQ simply remains dormant. This graceful fallback means that MoMQ-aware clients can connect to legacy MoQ-only relays without special handling. The client includes `ENABLE_MOMQ` in every session, and only relays that also advertise it will activate the extension.

Parameter Key (i) = `0x10`	Parameter Length (i)	Parameter Value (i) = `0x01`

Figure 3.3: Wire format of the `ENABLE_MOMQ` setup parameter. Both endpoints include this in their setup message to activate MoMQ.

3.3.2. PATH_STATE_REPORT Message

`PATH_STATE_REPORT` (type `0x52`) in the format shown in Figure 3.4 is sent by the relay to the subscriber to convey available paths and their labels. And Table 3.1 defines each field.

The relay MUST send a `PATH_STATE_REPORT` after MoMQ negotiation, when paths are added or removed, and when path status changes. The relay SHOULD rate-limit reports to at most one per smoothed RTT to avoid control-message flooding.

The rate-limiting recommendation needs further explanation. Path state can change rapidly, for example, during a Starlink reconfiguration, the satellite path's RTT may fluctuate on a millisecond timescale. Sending a report for every fluctuation would flood the control channel with stale information, since each report is outdated by the time the subscriber processes it and potentially updates rules. Rate-limiting to one report per smoothed RTT ensures that the subscriber sees a consistent view of path state at the timescale relevant to scheduling decisions. If the subscriber needs finer-grained path monitoring, it can observe transport-layer signals (such as ACK delays) directly without relying on MoMQ control messages.

3.3.3. PATH_LABEL_UPDATE Message

`PATH_LABEL_UPDATE` (type `0x53`) is sent by the subscriber to assign labels to paths.

Subscriber-assigned labels override relay-assigned labels with the same key, and the relay includes them in subsequent `PATH_STATE_REPORT` messages. A `PATH_LABEL_UPDATE` referencing an unknown Path ID is a Protocol Violation.

PATH_STATE_REPORT	
Sequence Number	(i)
Path Count	(i)
Path State Entry	(..) . . .

Path State Entry	
Path ID	(i)
Path Status	(8)
Label Count	(i)
Label	(..) . . .

Figure 3.4: Wire format of `PATH_STATE_REPORT` and its `Path State Entry` structure.

PATH_LABEL_UPDATE	
Path ID	(i)
Label Count	(i)
Label	(..) . . .

Label	
Key Length	(i)
Key	(..)
Value Length	(i)
Value	(..)

Figure 3.5: Wire format of `PATH_LABEL_UPDATE` and its `Label` structure.

Labels are general-purpose. Some labels are purely descriptive and participate only in rule evaluation (e.g., `cost_class`, `link_type`). Others may carry operational semantics that the relay acts on directly, for example, reconfiguring the congestion control algorithm on the path. Implementations define which label keys the relay interprets operationally, the wire format anyway is identical in both cases.

The distinction between descriptive and operational labels is important for understanding the scope of subscriber influence. A descriptive label like `cost_class=metered` affects scheduling only through rules that reference it. Therefore, if no rule mentions `cost_class`, the label actually has no effect. An operational label like `cca=leocc` may cause the relay to change the congestion control algorithm on that path, which has immediate transport-level consequences regardless of rules. Relay implementations should document which label keys they treat as operational and validate subscriber labels against an allowlist to prevent unintended configuration changes.

3.4. Object-to-Path Mapping Rules

This section defines MoMQ rules that map Object metadata to delivery preferences. The design follows a match-action paradigm conceptually similar to SDN flow rules [26]. A rule includes "match entries" that select Objects by metadata and "action entries" that express delivery preferences. Rules are session-scoped, advisory, and reference labels instead of path identifiers.

3.4.1. PATH_MAPPING_RULE Message

`PATH_MAPPING_RULE` (type `0x50`) is sent by the subscriber to install or remove rules. Figure 3.6 shows the wire format. Table 3.2 defines each field.

`INSTALL` creates a new rule or replaces an existing rule with the same Rule ID. `REMOVE` deletes the rule by ID. For `REMOVE`, Match Count and Action Count MUST be zero. All rules are discarded when the session terminates.

3.4.2. Match Operators

Table 3.3 defines the available match operators. The operator set is minimal to ensure that all relay implementations can support full matching capability.

Table 3.1: `PATH_STATE_REPORT` field definitions.

Field	Encoding	Description
Sequence Number	varint	Monotonically increasing; the subscriber MUST discard reports with a sequence number ≤ the most recently processed one
Path Count	varint	Number of Path State Entry structures that follow
Path ID	varint	The MPQUIC path identifier; both endpoints know this from the transport layer
Path Status	8-bit	ACTIVE (0x00), DEGRADED (0x01), or UNAVAILABLE (0x02)
Labels	variable	Key-value pairs describing path properties, including both relay-assigned and subscriber-assigned labels currently in effect

PATH_MAPPING_RULE	
Rule ID	(i)
Operation	(8)
Match Count	(i)
Match Entry	(..) . . .
Action Count	(i)
Action Entry	(..) . . .

Match Entry	
Key Length	(i)
Key	(..)
Operator	(8)
Value Length	(i)
Value	(..)

Action Entry	
Action Type	(8)
Params Length	(i)
Action Params	(..)

Figure 3.6: Wire format of `PATH_MAPPING_RULE` and its constituent structures. Each Action Entry includes a `Params Length` field that allows parsers to skip unknown action types.

Table 3.2: `PATH_MAPPING_RULE` message field definitions.

Field	Encoding	Description
Rule ID	varint	Non-zero identifier scoped to this session
Operation	8-bit	INSTALL (0x00) or REMOVE (0x01)
Match Count	varint	Number of Match Entry structures; zero for catch-all
Match Entry	variable	Key-operator-value condition over Object metadata
Action Count	varint	Number of Action Entry structures
Action Entry	variable	Delivery preference for matched Objects

Each rule is intentionally limited to a conjunction of metadata conditions, while disjunction is expressed by installing multiple rules. This AND-within-a-rule, OR-across-rules structure keeps evaluation simple while remaining expressive enough for the scheduling policies described in Chapter 4.

Multiple match entries within a rule are combined with AND logic, which means all entries must succeed for the rule to match. A key absent from the Object causes the match entry to evaluate to false (except `EXISTS`, which checks presence). Zero match entries produce a catch-all rule that matches every Object. OR logic is expressed by installing multiple rules, for example, if an application wants to apply the same action to both IDR frames and I-frames, it installs two rules, one matching `frame_type=IDR` and another matching `frame_type=I`, each with the same action entries.

The operator set is deliberately minimal. Only two operators might appear restrictive compared to SDN flow tables. This constraint is intentional because every relay implementation must support full matching capability, and complex operators increase the attack surface for denial-of-service via computationally expensive match evaluation. In practice, the two operators cover the most common scheduling patterns. `EQUALS` handles categorical distinctions (frame type, layer index, media type) that form the basis of almost all video-aware scheduling. `EXISTS` handles presence-based routing, such as directing all Objects that carry a `depends_on` field to a co-location action regardless of the specific dependency target.

More complex predicates can be expressed through rule composition. For instance, to differentiate between “base-layer IDR frames” and “enhancement-layer IDR frames,” a subscriber installs two rules, one matching both `frame_type=IDR` and `temporal_layer=0`, the other matching `frame_type=IDR` and `temporal_layer=1`. This uses two simple rules rather than a general-purpose expression evaluator.

3.4.3. Action Types

Table 3.4 defines the action vocabulary.

`PRIORITY` governs scheduling order among streams within the session, higher-priority streams are served first under contention. Default is 0. In a video conferencing scenario, an application might assign

Table 3.3: Match operators for metadata comparison.

Value	Name	Semantics
0x00	EQUALS	Exact byte-for-byte match between the metadata value and the specified value
0x01	EXISTS	The metadata key is present in the Object (Value field ignored)

Table 3.4: Action types and their parameters for expressing delivery preferences.

Type	Name	Parameters	Description
0x01	PRIORITY	priority (i)	Scheduling urgency; higher values are more urgent
0x02	BALANCING	mode (8)	0x00 = SINGLE_PATH, 0x01 = MULTI_PATH
0x03	PATH_PREFERENCE	key_len (i), key (..), value_len (i), value (..)	Prefer paths whose labels match the specified key-value pair
0x04	PATH_AFFINITY	ref_key_len (i), ref_key (..)	Co-locate with a previously forwarded Object identified by the named metadata key

priority 3 to IDR frames, priority 2 to base-layer P-frames, priority 1 to enhancement-layer frames, and priority 0 to audio which is small enough that queuing delay rarely matters. Under congestion, the relay scheduler dequeues priority-3 Objects first, ensuring that decoder-critical data reaches the subscriber before enhancement data competes for the same bandwidth. Without this signal, the MPQUIC scheduler treats all QUIC streams equally and may interleave a large enhancement-layer frame ahead of a small but critical base-layer frame.

`BALANCING` constrains path distribution, `SINGLE_PATH` forces all data of a stream onto one path to avoid cross-path reordering, while `MULTI_PATH` permits distribution across paths for throughput. Default is `SINGLE_PATH`. The default reflects a conservative design choice, cross-path reordering is the primary source of application-level latency degradation in the measurement study (Section 4.3), so the safe default avoids it. Applications that can tolerate reordering like bulk file transfers and pre-buffered video segments may prefer `MULTI_PATH` for better aggregate throughput.

`PATH_PREFERENCE` prefers paths whose labels contain the specified key-value pair, if no path matches or the matching path is unhealthy, the relay falls back to default selection. This fallback behavior is critical, it means that a rule preferring `cost_class=free` does not block data delivery when the free path is temporarily unavailable. The relay simply routes to the best available alternative, and when the preferred path recovers, subsequent Objects return to it automatically.

`PATH_AFFINITY` co-locates related Objects on the same path, the relay consults a bounded history of recent Object-to-path assignments to find the path used for a previously forwarded Object whose identifier matches this Object's metadata value for the named key. And if the reference is not found, the relay uses default path selection. The history is bounded to prevent unbounded memory growth. In practice, one GOP's worth of entries (typically 50–100 Objects) is sufficient for SVC dependency tracking. The relay evicts oldest entries when the history reaches its capacity.

All actions are advisory. The MPQUIC scheduler retains final authority over path selection based on transport-layer metrics and local policy. When multiple rules match an Object, their actions are merged according to the conflict resolution policy below.

3.4.4. Conflict Resolution

When multiple rules match an Object, actions are merged per dimension according to Table 3.5.

Table 3.5: Conflict resolution when multiple rules match an Object.

Dimension	Resolution
PRIORITY	Maximum value across matching rules
BALANCING	SINGLE_PATH if any rule requests it
PATH_PREFERENCE	Union of all preferences; relay resolves by local policy
PATH_AFFINITY	First matching rule takes precedence

The resolution policy is designed to be both safe and predictable. Consider three concrete scenarios. First, suppose a subscriber installs Rule A that assigns priority 2 to all IDR frames and Rule B that assigns priority 1 to all base-layer frames. An IDR frame at base layer matches both rules, therefore, the merged priority is $\max(2, 1) = 2$. Taking the maximum ensures that the most urgent classification wins because an Object cannot be "downgraded" by a less specific rule.

Second, suppose Rule A sets `MULTI_PATH` for high-throughput enhancement layers, while Rule B sets `SINGLE_PATH` for all video frames (as a safety default). A Layer 1 enhancement frame matches both rules, the merged balancing mode is `SINGLE_PATH` because the conservative option takes precedence. This prevents a permissive rule from accidentally introducing cross-path reordering.

Third, suppose Rule A prefers `cost_class=free` and Rule B prefers `link_type=satellite`. The merged preference is the union of both constraints, and the relay selects from paths that satisfy as many preferences as possible according to local scoring policy. If only one path satisfies both, it is chosen; if no single path satisfies all preferences, the relay falls back to the best partial match.

3.4.5. PATH_MAPPING_RESULT Message

The relay responds to each `PATH_MAPPING_RULE` with `PATH_MAPPING_RESULT` (type `0x51`) as shown in Figure 3.7.

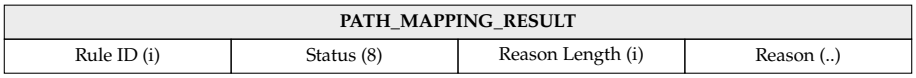

Figure 3.7: Wire format of `PATH_MAPPING_RESULT`.

Table 3.6 defines the status codes.

Table 3.6: `PATH_MAPPING_RESULT` status codes.

Code	Name	Description
0x00	OK	Rule accepted and installed
0x01	REJECTED	Rejected by relay policy
0x02	NOT_AUTHORIZED	Sender lacks permission for this operation
0x03	INVALID_RULE	Malformed rule structure
0x04	NOT_FOUND	Rule ID not found (for REMOVE operations)

3.5. Object Metadata

Mapping rules operate on Object metadata, namely key-value pairs attached to MoQ Objects. Metadata lets applications express Object semantics for relay-side matching without payload inspection, extending the per-object addressing model of MoQT [30] with application-defined attributes.

3.5.1. Metadata Model

Each Object may carry zero or more metadata entries. Keys are case-sensitive UTF-8 strings, and values are matched as byte strings. Metadata is per-Object, usually set by the publisher, and treated as immutable after Object creation. This keeps separation of concerns clear, publishers define semantics while relays perform mechanical matching and action application.

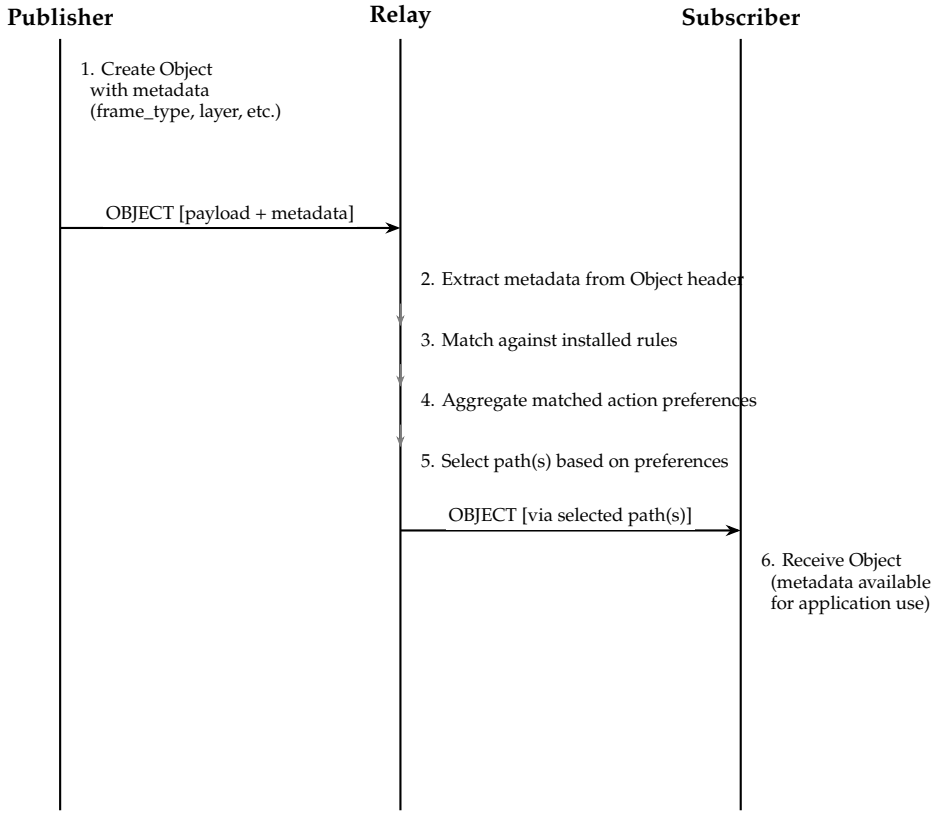

Figure 3.8: Metadata flow from publisher through relay to subscriber. The relay uses metadata for rule matching but does not interpret its semantic meaning. Metadata is preserved for subscriber applications.

3.5.2. Metadata Attachment and Flow

Figure 3.8 illustrates how metadata flows through the MoMQ system, from Object creation at the publisher through rule matching at the relay to delivery to the subscriber.

The relay role is mechanical: extract metadata, match rules, merge actions, and schedule paths. Metadata is forwarded unchanged, so subscriber-side logic can reuse publisher annotations.

3.5.3. Well-Known Metadata Keys

For interoperability, MoMQ defines well-known keys for video use cases (Table 3.7).

`frame_type` is central because it captures video dependency criticality. In practice, importance is typically IDR > I > P > B, which provides a natural basis for priority and scheduling policy. `temporal_layer` and `spatial_layer` support SVC adaptation, base layers are usually prioritized while enhancement layers can be degraded first under congestion. Keys include `track_id` and `group_id`; `object_id` aligns with MoQT hierarchy and enables rules scoped to tracks/groups/positions.

3.5.4. Application-Defined Metadata

Applications may define custom keys beyond the well-known set to support domain-specific scheduling policies. For example, a conferencing application might define a `speaker_active` key that marks Objects belonging to the currently speaking participant, enabling the relay to prioritize active speakers without understanding the conferencing protocol. A cloud gaming application might define `input_frame` to distinguish frames generated in response to player input (latency-critical) from background animation frames (delay-tolerant). A live sports broadcast might define `camera_id` to differentiate feeds from different camera angles, allowing the relay to prioritize the primary feed.

Relays treat custom keys exactly like well-known keys. The relay does not validate the semantic meaning of custom keys or enforce any naming convention. Missing keys cause non-match, not protocol error.

Table 3.7: Well-known metadata keys for video streaming applications.

Key	Value Type	Description
frame_type	string	Video frame type: “IDR” (instantaneous decoder refresh), “I” (intra-coded), “P” (predictive), or “B” (bidirectional)
temporal_layer	integer	Temporal layer identifier (0 = base layer, higher = enhancement)
spatial_layer	integer	Spatial layer identifier (0 = lowest resolution, higher = higher resolution)
track_id	integer	Identifier of the track containing this Object
group_id	integer	Identifier of the group containing this Object
object_id	integer	Identifier of this Object within its group
media_type	string	Media type: “video”, “audio”, “data”
codec	string	Codec identifier: “h264”, “h265”, “av1”, “vp9”, “opus”

This means that a rule referencing a custom key that some Objects do not carry will simply not match those Objects, and delivery proceeds normally through lower-priority rules or the default path.

3.5.5. Object Structure with Metadata

Figure 3.9 illustrates the logical structure of a MoQ Object with attached metadata, showing how metadata sits between the header and payload.

Separating metadata from payload enables lightweight scheduling decisions and works with encrypted payloads, since relays can process metadata without parsing media content.

3.5.6. Metadata and Privacy Considerations

Metadata can expose structure (for example, frame roles, layer patterns, or speaker/activity hints). Applications should balance scheduling benefit against exposure by minimizing fields, coarsening values, or omitting metadata when relay guidance is unnecessary.

The major concern is that metadata must be visible to the relay for rule matching, but visibility implies exposure. A relay that processes a stream of Objects with `frame_type` metadata can reconstruct the GOP structure of the video stream, e.g., the periodic pattern of IDR frames, the ratio of P-frames to enhancement frames, and indirectly the encoding bitrate. For many deployments this is acceptable, for example, the relay is operated by the same entity that provides the media service, and the scheduling benefit outweighs the information exposure. For deployments where the relay is operated by a

MoQ Object				
Object Header				
Track ID	Group ID	Object ID	Priority	Status
Object Metadata (MoMQ extension)				
Key: "frame_type"		Value: "IDR"		
Key: "temporal_layer"		Value: "0"		
Key: "media_type"		Value: "video"		
Key: "codec"		Value: "h264"		
Object Payload				
[Encoded media data (may be encrypted). . .]				

Figure 3.9: Logical structure of a MoQ Object showing the header, metadata key-value pairs, and payload. Metadata enables rule-based scheduling without payload inspection.

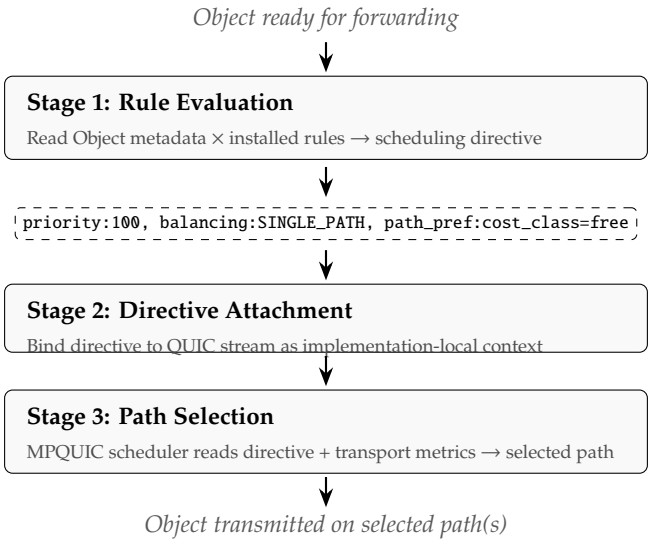

Figure 3.10: Per-Object scheduling pipeline. The relay evaluates rules against Object metadata to produce a scheduling directive, attaches it to the QUIC stream, and the MPQUIC scheduler reads the directive alongside transport metrics to select a path.

third party, applications should consider coarsening metadata. Reducing `frame_type` to a binary `is_keyframe` flag, or omitting `temporal_layer` when only frame-type-based scheduling is needed, limits the information available to the relay while preserving the scheduling policies that matter most.

3.5.7. Metadata Encoding Considerations

Metadata should remain compact, but readability is often more valuable than micro-optimizing a few bytes. In practice, include only fields needed by active rules. Metadata keys and values are encoded as UTF-8 strings in the wire format, which favors human readability during development and debugging over byte-level compactness. For production deployments where per-Object overhead matters, applications can use short key names and compact value representations. The overhead of metadata is typically small relative to media payload sizes. A video frame carrying four metadata key-value pairs adds roughly 40–80 bytes to an Object that may contain 5–230 KB of encoded video data.

3.6. Cross-Layer Scheduling Pipeline

This section defines the relay-side scheduling pipeline that connects MoQT Object semantics to MPQUIC path selection. The pipeline preserves MoQT's application-agnostic forwarding [30] while adding rule-driven path guidance atop MPQUIC's per-path transport state [12]. The pipeline executes in three stages each time an Object is forwarded to the subscriber (Figure 3.10).

3.6.1. Stage 1: Rule Evaluation

The relay reads the Object's metadata and evaluates it against all rules installed by this subscriber (Section 3.4). Each matching rule contributes actions, namely a priority value, a balancing constraint, a path preference by label, or an affinity with a previously forwarded Object. When multiple rules match, actions are merged by the conflict resolution policy (Section 3.4). The result is a *scheduling directive*, which is a flat record of resolved delivery preferences for this Object.

If no rules match, the directive uses defaults, e.g., priority 0, `SINGLE_PATH` balancing, no path preference, no affinity. The relay performs metadata extraction without payload parsing. Repeated metadata patterns may be cached for efficiency.

Table 3.8: Example scheduling decisions given path state (Path 0: 15 ms RTT, labeled `cost_class=free`; Path 1: 45 ms RTT, labeled `cost_class=metered`).

Object	Directive	Scheduling Decision
IDR frame (230 KB)	priority=100, balancing=SINGLE_PATH, path_pref: cost_class=free	High priority, schedule first. Prefer Path 0 (free). SINGLE_PATH: no splitting.
Base P-frame (10 KB)	priority=80, balancing=SINGLE_PATH, affinity: depends_on	Co-locate with dependency frame. SINGLE_PATH: select same path as reference.
Enhancement P-frame (8 KB)	priority=40, balancing=MULTI_PATH, path_pref: cost_class=free	Lower priority, schedule after critical Objects. May split across paths if beneficial.

3.6.2. Stage 2: Directive Attachment

The directive must cross the layer boundary to reach the MPQUIC scheduler. MoMQ achieves this by attaching the directive as implementation-local stream context to the QUIC stream that carries the Object. The directive is not wire data, it is scheduler-readable metadata bound to the stream within the transport implementation.

This is the direct link between application semantics and transport decisions. Before attachment, a QUIC stream is an opaque byte channel and the scheduler treats all streams identically. After attachment, the stream carries both data and intent. The scheduler does not need to parse MoQT framing, it just reads the directive directly from the stream context.

3.6.3. Stage 3: Path Selection

When the MPQUIC scheduler runs, it reads the directive from each stream that is ready to send and factors its contents into path scoring alongside transport metrics. `PRIORITY` governs scheduling order among streams. `BALANCING` constrains path distribution. `PATH_PREFERENCE` biases path scoring toward paths whose labels match the requested key-value pair. `PATH_AFFINITY` co-locates related Objects on the same path by consulting a history of recent Object-to-path assignments.

The directive is advisory. If the preferred path is congested or unavailable, the scheduler overrides the preference based on transport-level judgment. This preserves transport-layer autonomy [20] while giving the scheduler information it previously lacked.

Table 3.8 illustrates how different Object types with different directives result in different path assignments.

Worked example. Suppose an Object arrives with metadata `frame_type=IDR` and `temporal_layer=0`, and the subscriber has installed a rule that assigns `priority=100`, `balancing=SINGLE_PATH`, and `path_pref:cost_class=free` to all IDR frames. In Stage 1, the relay evaluates that metadata against the installed rules and produces exactly this directive. In Stage 2, the directive is attached to the QUIC stream carrying the Object. In Stage 3, if Path 0 is labeled `cost_class=free` with 15 ms RTT and Path 1 is labeled `cost_class=metered` with 45 ms RTT, the scheduler selects Path 0 and keeps the Object on a single path.

3.6.4. Default Behavior and Fallback

If no MoMQ session is negotiated, or no rules have been installed, the relay falls back to transport-layer scheduling and default MoQT forwarding behavior. This means the MPQUIC scheduler selects paths based solely on transport metrics including smoothed RTT, available congestion window and estimated loss rate without any application-level guidance. Objects are treated as opaque byte streams, and the scheduler makes decisions that are best for local transport performance but may be suboptimal for the application.

This fallback is not only a compatibility mechanism, but also an important safety feature. If the

subscriber's rule set is invalid, if the relay meets an unexpected error during rule evaluation, or if all installed rules are removed during a session, the system degrades smoothly to standard behavior rather than failing. The subscriber can always re-install rules to restore application-guided scheduling. This preserves full backward compatibility, a MoMQ-unaware relay operates identically to a standard MoQT relay, and a MoMQ-aware relay with no installed rules behaves the same way.

3.7. Security Considerations

MoMQ expands MoQT's control surface by allowing endpoints to influence scheduling through rules. This section summarizes the threat model and required safeguards.

3.7.1. Threat Model

We assume QUIC provides authenticated encryption and integrity protection [41], endpoints may be malicious or compromised, relays are trusted to enforce policy within their deployment domain, and multi-tenant isolation is required across sessions. The trust boundary in MoMQ differs from standard MoQT because the subscriber can influence relay scheduling by installing rules. In standard MoQT, the relay has full autonomy over forwarding; in MoMQ, the subscriber can shape, though not dictate, scheduling decisions. This makes the subscriber a potential adversary at the scheduling level, even though it is an authenticated participant at the transport level. The relay should therefore check and limit subscriber control to prevent abuse.

The main threats follow directly from the three actors that gain leverage once scheduling becomes rule-driven. A *malicious subscriber* can try to consume disproportionate resources or interfere with other sessions by installing bad rules. A *compromised relay* already sits on the forwarding path and could use rule processing to bias delivery or leak topology information. A *passive observer* cannot install rules, but may still infer content structure from visible metadata patterns and scheduling behavior. The rest of this section follows these actors through the corresponding safeguards: authorization and resource limits constrain subscriber abuse, advisory semantics and integrity assumptions bound relay behavior, and metadata minimization plus label indirection reduce what an observer can learn.

3.7.2. Authorization

Rule installation is privileged because it affects resource allocation. The main threats are unauthorized rule installation, rule injection across sessions, and cross-session interference. Relays MUST authenticate rule senders via session identity, enforce explicit authorization policy (subscriber-scoped, token-based, or equivalent), return `NOT_AUTHORIZED` for rejected operations, and keep rule state strictly session-scoped and isolated. MoMQ leverages the QUIC session identity established during TLS handshake for this purpose. Since each MoMQ session is bound to exactly one QUIC connection, the relay can clearly attribute every rule to its originating subscriber. The relay maintains a per-session rule store that is invisible to other sessions, ensuring that a subscriber cannot read, modify, or be affected by another subscriber's rules.

Authorization policy is deployment-specific. A relay in a managed enterprise environment might grant rule installation to all authenticated subscribers, while a relay serving untrusted public clients might require explicit authorization tokens issued by the application provider. The `NOT_AUTHORIZED` response code intentionally reveals only the fact that this operation was denied, not the detailed reason why it was denied, to avoid leaking policy details that could help an attacker probe the authorization boundary.

3.7.3. Resource Exhaustion

Rules consume memory, CPU, and bandwidth, and can be abused by rule flooding, high-complexity matching, excessive bandwidth use, or priority abuse. Relays SHOULD apply bounded resource controls, including caps on rules per session, match entries per rule, key/value lengths, and per-rule action counts, along with rule installation rate limiting and cross-session fairness/priority normalization. The resource limits in Table 3.9 are designed to accommodate legitimate applications while preventing abuse. The 100-rule cap per session is based on the observation that even complex applications (multi-layer SVC with reconfiguration-aware scheduling) require fewer than 10 rules. 100 provides headroom for

Table 3.9: Recommended resource limits for MoMQ relay implementations.

Resource	Suggested Limit	Rationale
Rules per session	100	Sufficient for complex applications
Match Entries per rule	10	Beyond this, rules become unwieldy
Key length	128 bytes	Keys should be short identifiers
Value length	1024 bytes	Values may contain paths or long IDs
Rule installation rate	10/second	Normal sessions need few rule changes
Actions per rule	20	Limits scheduling complexity

future use cases. The 10-entry match limit per rule reflects the maximum number of metadata fields that a single matching decision would reasonably inspect. The rate limit of 10 rule installations per second is sufficient for dynamic adaptation (responding to path changes with updated rules) while making rule flooding impractical as an attack vector.

When a relay reaches a resource limit, it SHOULD reject the offending operation with `REJECTED` status code rather than silently dropping rules or terminating the session. This allows the subscriber to receive clear feedback and adjust its behavior. For multi-tenant relays serving many concurrent subscribers, cross-session fairness requires an additional layer: the relay should normalize priorities across sessions so that one subscriber's priority escalation does not starve other sessions.

3.7.4. Privacy

Even with encrypted payloads, metadata and behavior can still leak information. This leakage can come from topology inference, content-structure hints, or rule inference. Rules MUST NOT carry explicit path identifiers. Responses MUST NOT disclose topology or path IDs. Errors SHOULD remain generic with respect to path internals. Applications should minimize metadata and use coarse values when possible. The label indirection layer provides a first line of defense because rules reference labels rather than path identifiers. This prevents a subscriber from learning the relay's internal path numbering or topology from the rule protocol alone. However, metadata attached to Objects can still reveal content structure. For example, a relay that observes a periodic pattern of `frame_type=IDR` followed by sequences of `frame_type=P` can infer the GOP structure and encoding parameters of the video stream, even without decrypting the payload. Similarly, `temporal_layer` values reveal the number of SVC layers and their relative frequency.

These leakages are inherent to the metadata-driven approach. The relay must see metadata to match rules, so any information visible to the relay is potentially exposed. Applications can mitigate this trade-off in several ways. First, metadata should include only the fields that active rules actually reference. Unused fields provide no scheduling benefit but increase the information exposed. Second, values can be coarsened. For instance, an application can report `frame_type=key` versus `frame_type=non-key` rather than the full IDR/I/P/B taxonomy if the scheduling policy only distinguishes key frames from others. Third, in scenarios where the relay is fully trusted (e.g., an operator-managed edge relay), privacy constraints can be relaxed in favor of richer scheduling.

3.7.5. Integrity

MoMQ relies on QUIC AEAD integrity. Implementations MUST preserve these guarantees and avoid unsafe shortcuts (for example, replay-prone handling for sensitive control paths).

3.8. Chapter Summary

This chapter has presented the complete design of MoMQ, an extension to the Media over QUIC Transport protocol [30] that enables application-guided scheduling of media Objects across multiple network paths. The design connects MoQT's object-oriented media model to Multipath QUIC's packet-level path scheduling [12], enabling content-aware delivery without compromising the relay architecture that makes MoQT scalable.

3.8.1. Design Contributions

The MoMQ design makes four key contributions to media transport over multipath networks. First, it introduces a minimal protocol extension: one setup parameter (`ENABLE_MOMQ`) and four control messages (`PATH_STATE_REPORT`, `PATH_LABEL_UPDATE`, `PATH_MAPPING_RULE`, and `PATH_MAPPING_RESULT`). This footprint reduces implementation complexity, enables incremental deployment, limits the attack surface, and allows the extension to fit into existing MoQT implementations without major refactoring. Second, it provides rule-based object scheduling with four action types, `PRIORITY`, `BALANCING`, `PATH_PREFERENCE`, and `PATH_AFFINITY`, that separate policy from mechanism and enable differentiated scheduling through a declarative API. Third, it introduces *path labels* as a layer of indirection between application intent and transport topology, rules express preferences over labels (such as `cost_class=free`) rather than explicit path identifiers, which preserves portability, privacy, relay autonomy, and adaptation as path conditions change. Fourth, it preserves MoQT's application-agnostic relay architecture, relays perform mechanical rule matching without interpreting metadata semantics. While this thesis evaluates SVC conferencing, the rule mechanism is application-agnostic and could apply to cloud gaming or live streaming.

3.8.2. Protocol Elements

Table 3.10 summarizes the protocol elements introduced by MoMQ and their roles in the design.

Table 3.10: Summary of MoMQ protocol elements.

Element	**Type**	**Purpose**
ENABLE_MOMQ (0x10)	Setup Parameter	Declare MoMQ capability; binary (present = enabled)
PATH_STATE_REPORT (0x52)	Control Message	Relay reports available paths, status, and labels to subscriber
PATH_LABEL_UPDATE (0x53)	Control Message	Subscriber assigns or overrides labels on paths
PATH_MAPPING_RULE (0x50)	Control Message	Install or remove Object-to-path scheduling rules
PATH_MAPPING_RESULT (0x51)	Control Message	Acknowledge rule operations with status codes
Match Entry	Rule Component	Specify metadata matching criteria (key with EQUALS or EXISTS operator)
Action Entry	Rule Component	Specify delivery preferences (priority, balancing, path preference, path affinity)
Path Label	Data Model	Key-value pair on a path; rules reference labels rather than path IDs
Object Metadata	Data Model	Key-value pairs attached to Objects describing their properties

3.8.3. Design Trade-offs

The MoMQ design reflects several conscious trade-offs. Simplicity is favored over expressiveness, the rule language uses AND-only logic and two match operators (`EQUALS`, `EXISTS`) so all relays can implement full matching capability, while complex policies can be expressed by composing simple rules. Advisory semantics are favored over mandatory requirements to preserve relay autonomy and enable graceful degradation under resource constraints. The balance between privacy and optimization is left to applications, since metadata improves scheduling but can expose content structure; path labels carry similar exposure risk. Finally, flexibility is favored over predictability, transport-agnostic rules work across diverse networks, but do not guarantee a specific path choice. Together, these choices provide general-purpose primitives while maintaining MoQT's simplicity and scalability.

4. Scalable Video Conferencing over MoMQ

This chapter studies how MoMQ can be used for SVC video conferencing over a Starlink LEO satellite link with a terrestrial backup path. It has two main goals. First, it shows that MoMQ's flexible rule framework can support the scheduling policies needed in a real application. Second, it demonstrates through measurement that transport-only multipath schedulers or single-path connections cannot solve the application-level problems in this scenario.

The structure is organized in a diagnosis-then-treatment style. Section 4.1 introduces the problem and identifies four scheduling problems that appear when SVC traffic runs over heterogeneous multipath networks. Section 4.2 measures these problems using controlled measurements over real infrastructure. Section 4.3 shows that transport-only multipath schedulers cannot solve them, which highlights the need for application-level support. Sections 4.4–4.5 develop scheduling mechanisms from the measurement findings and express them as declarative MoMQ rules following the framework of Chapter 3. Section 4.6 validates the contribution of each scheduling rule through a controlled simulation, Section 4.7 confirms these findings on the real Starlink testbed, and Section 4.8 tests whether the gains remain across different topologies by changing the subscriber and relay locations.

4.1. Problem Statement

Real-time video conferencing requires end-to-end latency below 150 ms for interactive use [19]. SVC encoding [37] improves network adaptability by allowing temporal layers to be dropped. This flexibility also creates a structural challenge: within each GOP, one IDR frame contains 15–30× more data than a P-frame, causing periodic traffic bursts that dominate tail latency even when average throughput is sufficient. When this bursty traffic pattern is sent over LEO satellite links [29], where periodic reconfigurations can cause brief but severe disruptions [40], four scheduling problems emerge that cannot be solved by single-path connections or transport-only multipath schedulers.

Problem 1: IDR head-of-line blocking. An IDR frame of 230 KB (Table 2.2) needs 2–5 congestion-window rounds to transmit. During these rounds, all following P-frames are queued behind the IDR frame, increasing their completion time from less than 20 ms to 100–300 ms. Because each GOP has only one IDR frame but up to 49 P-frames, one slow IDR frame cascades delay across the entire group. This happens due to the size asymmetry between frame types, and transport schedulers that treat all bytes equally cannot prevent it.

Problem 2: IDR completion-time inflation. LEO satellites experience periodic reconfigurations that interrupt the link for 50–170 ms. If an IDR frame is being sent during a reconfiguration, its delay can rise above 700 ms, which is far beyond a 150 ms latency target. Although reconfigurations are fairly rare (4 per minute on Starlink), their collision with IDR frames produces tail events that dominate the user-perceived quality.

Problem 3: Dependency-order violations. SVC's temporal layers have strict decoding dependencies [37], a Layer 1 frame cannot be decoded until its Layer 0 reference arrives. When a multipath scheduler splits dependent frames across paths with asymmetric RTTs (e.g., 15 ms WiFi vs. 45 ms Starlink), the receiver must buffer early-arriving frames until their dependencies arrive on the slower path. Transport schedulers have no visibility into these dependencies and thus cannot prevent the reordering.

Problem 4: Reactive secondary-path overuse. Transport-only schedulers optimize for latency or throughput without considering access cost. On a Starlink-plus-cellular setup, schedulers like MinRTT route 90% of traffic to the lower-RTT cellular path, reducing the value of flat-rate satellite service. As a result, the user pays metered cellular costs for only minor latency gains.

These four problems come from the same root cause that transport-layer schedulers do not have enough application-level context to tell the difference between an IDR frame, whose delay stalls an entire GOP, and an enhancement P-frame, whose loss affects a single decoded frame. Chapter 3 introduced

MoMQ's rule-based scheduling framework for exactly this kind of semantic mismatch. The rest of this chapter quantifies these problems through measurement (Section 4.2), demonstrates that transport-only schedulers fail to address them (Section 4.3), and derives an MoMQ rule set that resolves all four (Sections 4.4–4.7).

4.2. Measurement Study

4.2.1. Experimental Setup

Our testbed consists of a Starlink terminal with an eduroam WiFi connection located at the TU Delft campus in the Netherlands (52.0°N, 4.4°E), connected to a server in the Hetzner datacenter in Nuremberg, Germany. The Starlink terminal provides LEO satellite connectivity with throughput ranging from 50 to 100 Mbps and RTT varying between 25 and 80 ms depending on satellite position and reconfiguration state. The WiFi connection serves as the secondary path, emulating a cellular backup link.

We transmit a 100-second 1080p video encoded with SVC [37] using 1-second GOPs at 50 fps. The SVC encoding uses three temporal layers: Layer 0 carries base-layer frames essential for decoding; Layer 1 and Layer 2 carry enhancement frames that improve quality but can be dropped under congestion without breaking the decode chain. Each GOP begins with a single IDR (Instantaneous Decoder Refresh) frame that serves as a random access point, all subsequent frames in the GOP depend on it directly or transitively. The choice of 1-second GOPs reflects common conferencing practice that shorter GOPs provide faster recovery from packet loss but increase overall bitrate due to more frequent IDR frames, while longer GOPs are more efficient but amplify the impact of each IDR loss.

We test four congestion control algorithms, namely Cubic [17], BBRv3 [8], Copa [2], and LeoCC [23] (the latter only on Starlink). Cubic represents the baseline loss-based approach widely deployed in TCP and QUIC. BBRv3 is a model-based algorithm that estimates bandwidth and RTT to avoid buffer bloat. Copa is a delay-oriented algorithm that explicitly trades throughput for lower queueing delay, making it well-suited to latency-sensitive applications. LeoCC is specifically designed for LEO satellite links, using Kalman-filtered RTT estimation to separate reconfiguration-induced latency spikes from genuine congestion signals. All measurements are recorded with per-packet RTT timestamps at millisecond precision.

We evaluate three primary metrics, namely *minimum buffer length*, the smallest playback buffer ensuring stutter-free playback; *frame completion time* (FCT), the duration from first packet transmission to last packet acknowledgment for each frame; and *transmission rounds*, the number of congestion windows required to transmit a frame. Each configuration is measured over 10 runs, yielding 100 datapoints per CCA for buffer analysis (10 runs $\times$ 10 segments) and approximately 5 000 frames per CCA for FCT analysis. We report percentiles rather than means for tail-latency analysis, since a small number of extreme events, a single IDR-reconfiguration collision, can dominate the user experience while leaving the mean nearly unchanged.

4.2.2. WiFi Baseline

We first establish baseline performance over the controlled WiFi path. Across Cubic, BBRv3, and Copa, the minimum stutter-free buffer lies in the 300–450 ms range (Figure 4.1), far above a 150 ms interactive conferencing target. Copa achieves the lowest median (338.49 ms, σ = 34.43 ms, IQR = 50.39 ms); BBRv3 is intermediate (381.12 ms, σ = 51.42 ms, IQR = 54.90 ms); and Cubic is the worst and most volatile (410.78 ms, σ = 101.02 ms, IQR = 141.02 ms). All three CCAs exceed the interactive budget by at least 2×, confirming that burstiness is the bottleneck instead of average throughput.

The frame completion time distribution reveals a pronounced tail even on this controlled link (Figure 4.2). Per-CCA medians are modest (Cubic 31.7 ms, BBRv3 28.4 ms, Copa 25.2 ms), but the tail diverges sharply: P95 values reach 96.9/84.4/71.5 ms and P99 values reach 256.6/224.8/235.0 ms respectively. While 77–89% of frames complete within 50 ms, 3–5% exceed 100 ms, a significant fraction for real-time communication. All frames exceeding 200 ms are I-frames, and the first 3–5 P-frames following each I-frame are disproportionately delayed by head-of-line blocking during multi-round transmission.

The distribution of I-frame transmission rounds (Figure 4.3) explains why. Virtually no I-frames complete

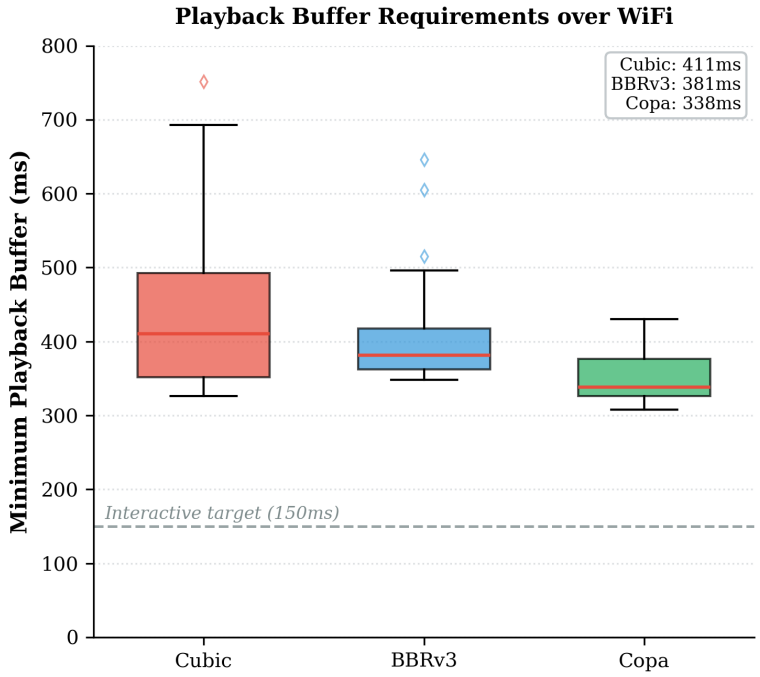

Figure 4.1: Minimum playback buffer requirements over WiFi (100 datapoints per CCA). Box plots show median (red line), interquartile range (box), and outliers (diamonds). All CCAs require 300–450 ms buffers, far exceeding the 150 ms interactive target.

within a single congestion window (0–8% across CCAs), reflecting the fundamental mismatch between I-frame size (230 KB) and typical cwnd. Copa concentrates 62.4% of I-frames in 2–3 rounds with only 2.6% requiring five or more; BBRv3 places 28.6% in round 2, 59.0% in rounds 3–4, and 11.4% at five or more; Cubic is worst, with 35.4% of I-frames requiring five or more rounds and only 8.8% completing in two, consistent with its high buffer variance. This multi-round transmission pattern is the root cause of the convoy effect: while an I-frame occupies the congestion window for 2–5 RTTs, all dependent P-frames are blocked behind it, inflating tail latency across the entire GOP.

4.2.3. Starlink Performance

Moving from the controlled WiFi baseline to the Starlink satellite path, we add LeoCC to the CCA comparison. Buffer requirements rise by 15–19% across all CCAs (Figure 4.4). Cubic degrades most (median 475.13 ms, σ = 133.67 ms, max 1306.47 ms), followed by BBRv3 (452.90 ms, σ = 102.35 ms, max 961.91 ms) and Copa (397.30 ms, σ = 86.74 ms, max 887.14 ms). LeoCC achieves the lowest Starlink buffer (371.55 ms, σ = 53.40 ms, max 723.49 ms), a 22% improvement over Cubic, yet still exceeds the 150 ms interactive target by 2.5×.

The tail is substantially worse. Median FCT increases modestly over WiFi (Cubic 42.3 ms, BBRv3 37.8 ms, Copa 34.5 ms, LeoCC 32.6 ms), reflecting Starlink's higher base RTT. P99 jumps to 335.3/310.3/276.4/258.4 ms and P99.9 reaches 757.5/712.3/681.1/541.7 ms respectively. For Cubic, BBRv3, and Copa, approximately 0.12% of frames exceed 500 ms, corresponding to 6–7 deadline misses per 2-minute session. LeoCC reduces this rate to 0.06%, yet tail events persist, confirming that the root cause is not purely congestion control but the collision between I-frame transmission and reconfiguration events.

4.2.4. Reconfiguration Analysis

To understand this collision effect, we analyze 5 998 frames sent over Starlink. Among them, 352 frames (5.9%) are delayed above 200 ms, showing a bimodal severity pattern (Figure 4.5). Severe delays (>400 ms, mean 497.2 ms) occur for 35 frames and happen only when I-frame transmission directly overlaps a reconfiguration window. We detect 7 such collisions across 8 detected events, and each cascades to 1–5 subsequent P-frames (20 total). The other 325 delayed frames fall in the moderate range

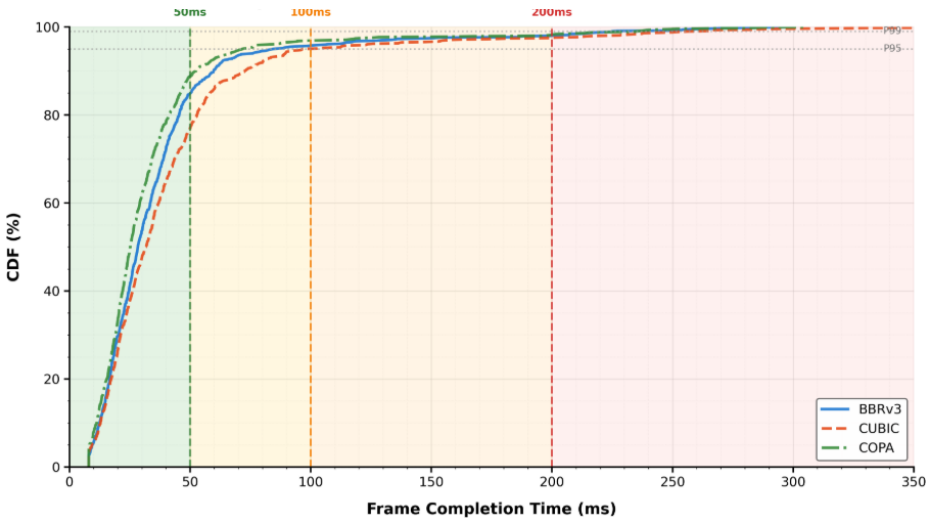

Figure 4.2: CDF of frame completion time over WiFi. Vertical dashed lines indicate 50, 100, and 200 ms thresholds. The tail is driven by I-frame transmission rounds, not by median-rate behavior.

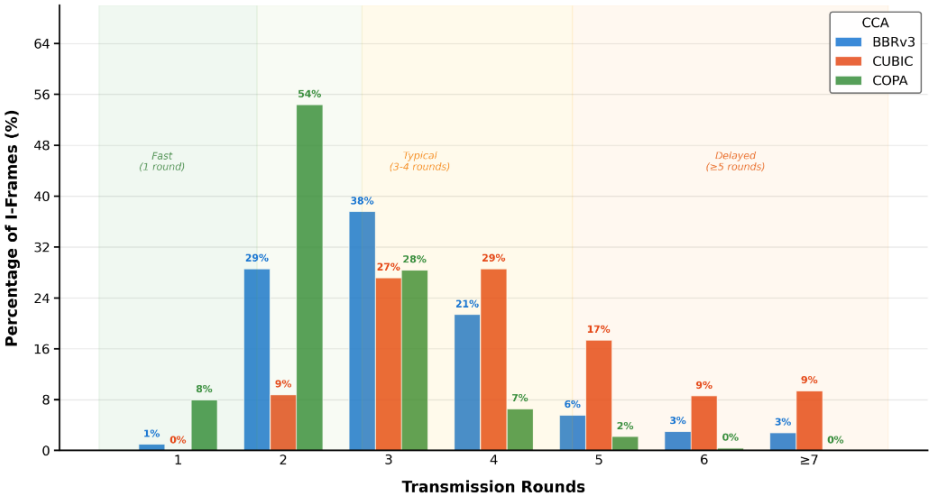

Figure 4.3: Distribution of I-frame transmission rounds over WiFi. Most I-frames require multiple congestion-window rounds, creating convoy effects for subsequent P-frames.

(200–400 ms), driven by inherent I-frame burstiness even when no reconfiguration happens.

The convoy effect spreads unevenly from each I-frame boundary. Statistical analysis across 100 GOPs shows that I-frames show the highest delayed ratio (14.8%). The first following P-frame at position +1 has a delay ratio of 7.2%, which then gradually decreases from +2 to +5 (6%→4%→2.5%→1.8%) and remains close to 1% from positions +6 to +25. In contrast, P-frames *preceding* the I-frame (positions −24 to −1) show uniformly low delay ratios (0.9%), confirming that blocking is unidirectional. Dropping delayed I-frames is not an option, each I-frame anchors an entire 1-second GOP of 50 frames, so even 1% I-frame loss over a 30-minute call translates to roughly 18 decoder stalls.

These observations raise a natural question, can reconfiguration events be predicted and avoided? We investigate this by analyzing 115 validated inter-event intervals across 20 runs (Figure 4.6). Starlink exhibits a highly predictable 15-second reconfiguration cycle with sub-millisecond mean timing accuracy. The mean deviation from the expected period is 0.5 ms, and 92.2% of intervals fall within ±20 ms (σ = 13.2 ms); all intervals lie within ±50 ms. Of these, 86.1% follow the single base period (15 s), 9.6% span 2× (30 s, one skipped reconfiguration), and 4.3% span 3× (45 s, two skipped).

Event duration is bounded (Figure 4.7). Across 103 detected events during 30 minutes of continuous

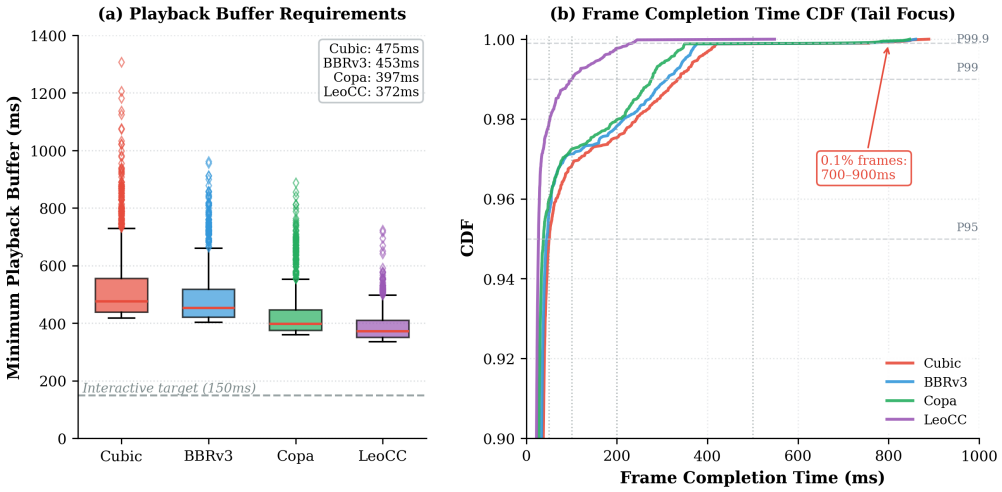

Figure 4.4: Starlink performance analysis. (a) Minimum playback buffer requirements across four CCAs. (b) CDF of frame completion times with tail focus. Reconfiguration events enlarge the tail substantially compared to the WiFi baseline.

Table 4.1: Summary of measurement findings and their implications for scheduling design.

#	Finding	Implication
F1	I-frames block P-frames during multi-round transmission	Need interleaving scheduler
F2	0.1% of frames hit reconfiguration → 700–900 ms delay	Cannot ignore rare events
F3	Reconfiguration timing highly predictable (σ = 13.2 ms, 92% within ±20 ms)	Can proactively avoid danger windows
F4	Reconfiguration duration is brief (median 58 ms, 66% within 30–70 ms)	Short avoidance window suffices
F5	I-frame loss affects entire GOP (50 frames)	Cannot drop; must ensure delivery

1 ms-rate probing, durations range from 22 ms to 172 ms, with median 58 ms, mean 67 ms, σ = 31.4 ms, and IQR 49–81 ms. The majority (66%) complete within 30–70 ms, with a pronounced peak in the 40–70 ms core range. Events exceeding 120 ms are infrequent (7.8%). A conservative 100 ms deferral window accommodates 88% of all events, and the wall-clock schedule (reconfigurations at the 12th, 27th, 42nd, and 57th second of each minute) is deterministic enough that proactive avoidance is feasible.

4.2.5. Summary of Findings

Table 4.1 summarizes the measurement results into five findings that guide the rest of this chapter.

Single-path transport is fundamentally inadequate for SVC-based real-time communication. On WiFi, the 23:1 size ratio between I-frames and P-frames causes unavoidable head-of-line blocking regardless of CCA choice (F1). On Starlink, periodic reconfiguration events worsen this with catastrophic delays (F2), and each affected I-frame disrupts an entire 1-second GOP (F5). Even LeoCC cannot eliminate these tail events because the root cause is frame-level scheduling, not congestion control. Reconfiguration timing is nevertheless both predictable (F3) and brief (F4), suggesting that proactive avoidance via a backup path could be effective, provided the transport layer knows *which* frames need protection.

4.3. Transport-Only Multipath Scheduling

These limitations motivate an evaluation of Multipath QUIC [12] as a source of path diversity. The central question is whether two paths can jointly provide low latency and resilience by assigning one path to

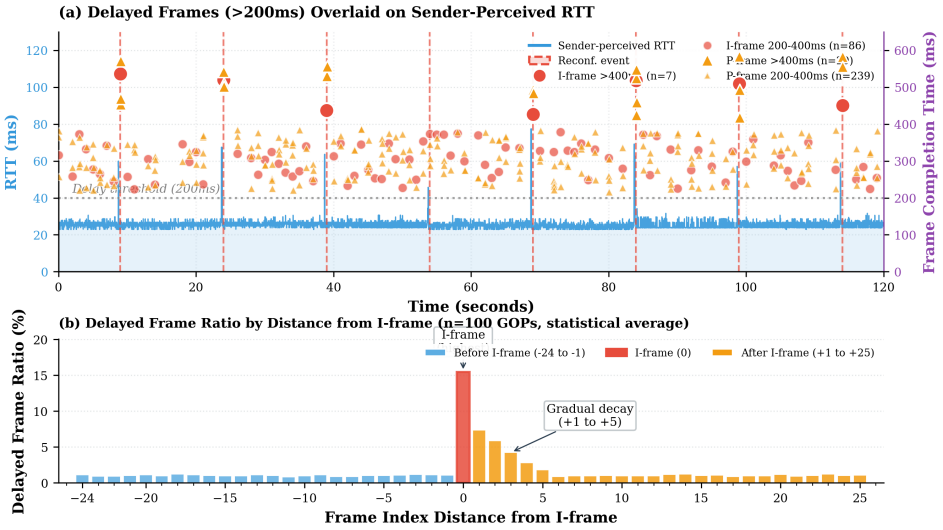

Figure 4.5: Delayed frames overlaid on RTT traces. (a) Larger markers indicate severe delays (>400 ms) from reconfiguration collisions; smaller markers indicate moderate delays (200–400 ms). (b) Distribution of delayed-frame ratio by distance from I-frame boundary.

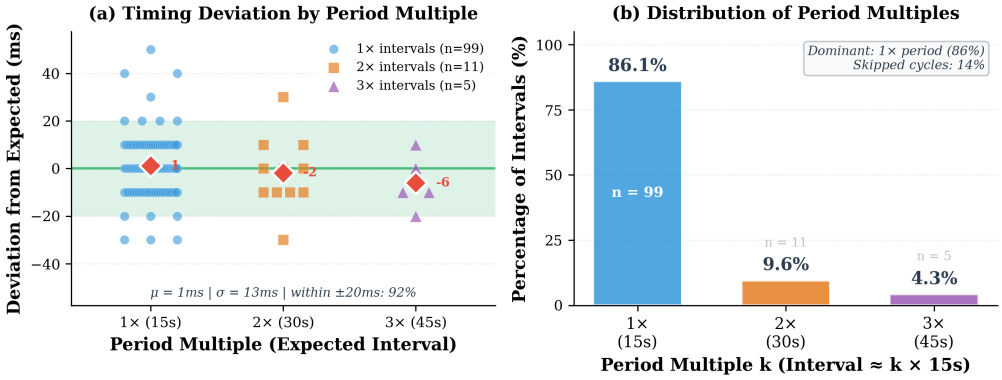

Figure 4.6: Starlink reconfiguration event analysis across 20 runs (n = 115 intervals). (a) Timing jitter distribution; the green band marks ±20 ms. (b) Distribution of intervals as multiples of the 15-second base period.

latency-sensitive delivery and the other to capacity or redundancy. We evaluate four representative MPQUIC schedulers over the same testbed with Starlink as the primary path and WiFi as the backup. Each scheduler embodies a different policy for using multiple paths. MinRTT routes each packet to the lowest-RTT path, optimizing for latency at the expense of path balance. Round-Robin distributes packets evenly across paths, ignoring RTT asymmetry. BLEST [14] estimates whether the slower path would cause head-of-line blocking and avoids it when it would, attempting to capture the benefits of both paths. Redundant transmits every packet on both paths, trading bandwidth for reliability.

The results (Figure 4.8, Table 4.2) show a fundamental failure. MinRTT strongly biases toward WiFi (89.7% of traffic); when a 230 KB I-frame fills the WiFi congestion window (82 KB), overflow packets spill onto Starlink (45 ms RTT). The receiver must wait for these slower packets before the I-frame can be decoded, producing cross-path head-of-line blocking that is *worse* than single-path WiFi (360.18 ms vs. 338.49 ms). Round-Robin distributes the roughly 161 packets of each I-frame cyclically, so roughly half arrive via WiFi at 15 ms while the rest arrive via Starlink at 45 ms, causing a systematic 30 ms per-round reordering penalty that accumulates across 3+ rounds, making it the worst configuration overall (445.31 ms, σ = 82.41 ms). BLEST avoids Starlink when it estimates blocking would occur, effectively behaving as a smarter single-path WiFi scheduler; its modest improvement (332.37 ms) comes at 91.8% WiFi usage. Redundant duplicates every packet on both paths, reducing variance (σ = 27.14 ms) at 2× total bandwidth, yet its median (334.23 ms) merely matches WiFi.

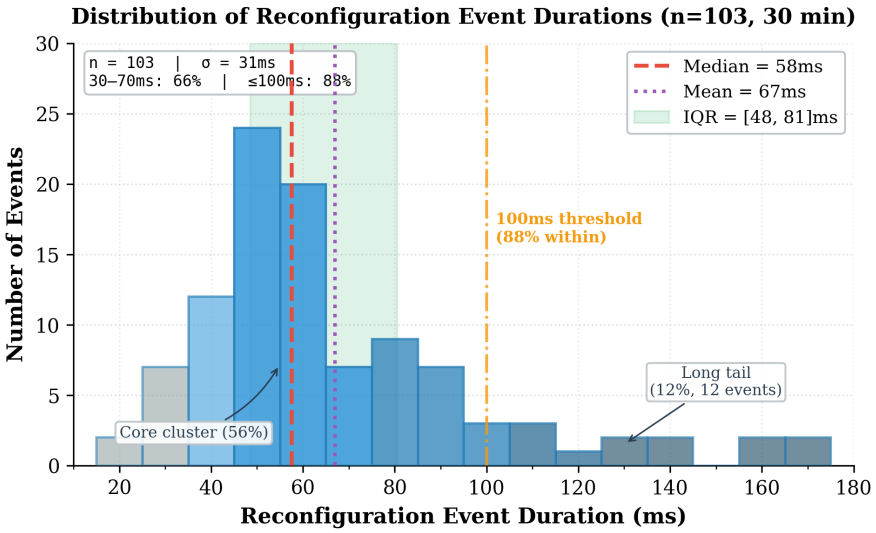

Figure 4.7: Reconfiguration event duration distribution (n = 103). The dashed red line indicates the median (58 ms); the green shaded region is the IQR (49–81 ms). A 100 ms deferral window accommodates 88% of events.

Table 4.2: Playback buffer comparison: single-path baselines and transport-only multipath schedulers.

Configuration	Median Buffer (ms)	Std Dev (ms)	Backup Usage
Copa / WiFi (single-path)	338.5	34.5	100%
LeoCC / Starlink (single-path)	367.6	53.4	0%
MPQUIC MinRTT	360.2	53.9	89.7%
MPQUIC Round-Robin	445.3	82.4	47.6%
MPQUIC BLEST	332.4	36.2	91.8%
MPQUIC Redundant	334.2	27.1	100% (dup.)

The central trade-off failure is clear, schedulers that keep latency near the best single-path values route 90–100% of traffic to the metered backup path without significantly improving tail performance, while the one that balances paths (Round-Robin) performs *worse* than any single-path baseline. The underlying problem is a semantic gap. Transport-layer schedulers cannot distinguish a 230 KB I-frame whose delay stalls an entire GOP from a 10 KB enhancement P-frame whose loss causes only localized degradation. Without this knowledge, they cannot interleave P-frames during multi-round I-frame transmission to prevent head-of-line blocking (F1), cannot avoid the Starlink path during predicted reconfiguration windows (F2–F4), cannot co-locate dependent enhancement-layer frames on the same path as their references to avoid cross-path reordering, and cannot enforce a cost-sensitive default that keeps bulk traffic on the flat-rate primary path.

This failure identifies the information required for effective scheduling. MinRTT and BLEST make locally optimal decisions given what they can observe (RTT, congestion window), yet the information they lack (frame type, frame dependencies, reconfiguration timing, access cost) is precisely the information that determines application-level quality. Round-Robin fails for the most direct reason that it treats all packets identically and therefore distributes dependent data across paths with different delays. Even the more sophisticated schedulers nevertheless fail because sophistication in the transport dimension does not compensate for blindness in the application dimension. This observation motivates the MoMQ approach that rather than building a smarter transport scheduler, add a channel through which application semantics can reach the existing scheduler.

4.4. Scheduling Mechanisms

Closing this semantic gap requires four mechanisms, each targeting a distinct finding from the measurement study. Prior work on tighter codec-transport integration [16] and WebRTC congestion

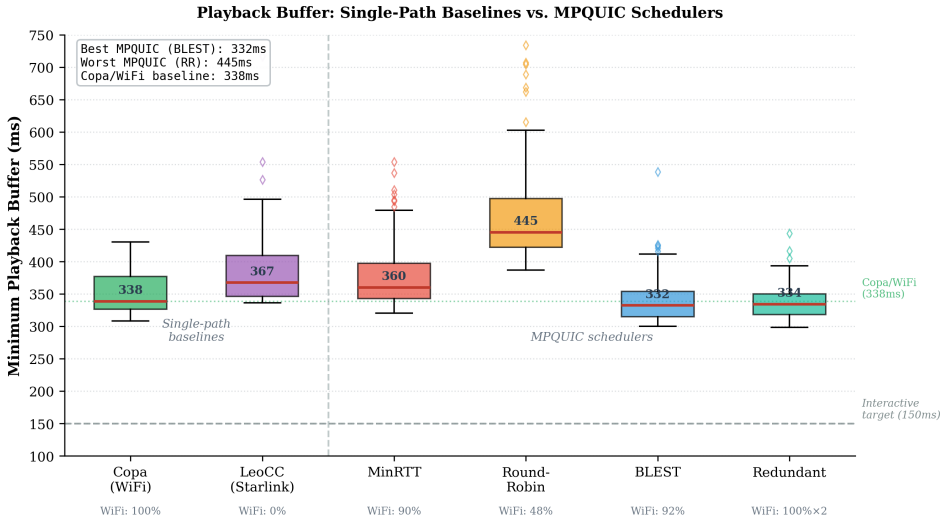

Figure 4.8: Playback buffer requirements: single-path baselines versus four MPQUIC schedulers. The dashed green line is the Copa/WiFi median (338 ms); the dashed gray line is the 150 ms interactive target. No transport-only scheduler improves upon single-path WiFi.

control [9] addresses parts of the problem in single-path settings. Multipath introduces additional scheduling dimensions that these approaches do not cover.

The mechanisms are designed to be composable, each one addresses one problem. Budget-based interleaving operates within a single congestion-window cycle, reconfiguration avoidance operates on a wall-clock schedule, dependency co-location operates on frame relationships, and cost preference operates on path classification.

Budget-based P-frame interleaving addresses the convoy effect (F1). During I-frame transmission spanning k RTT cycles, there exists slack capacity. The difference between the total data the congestion window can carry and the I-frame size is:

$$B_{\text{budget}} = \left\lceil \frac{S_I}{cwnd_{\text{cons}}} \right\rceil \cdot cwnd_{\text{cons}} - S_I, \quad \text{where } k = \left\lceil \frac{S_I}{cwnd_{\text{cons}}} \right\rceil \tag{4.1}$$

This slack can accommodate P-frame packets without noticeably delaying I-frame completion. The key insight is that P-frames can "cut in line," interleaving their packets within the idle capacity of each congestion-window round. To ensure interleaving does not harm I-frame delivery, we derive $cwnd_{\text{cons}}$ from LeoCC's conservative (Kalman-filtered) bandwidth estimate $\widehat{BW}_m$ and minimum filtered RTT $bRTT_l$: $cwnd_{\text{cons}} = \widehat{BW}_m \times bRTT_l$. For a typical case with $S_I = 230$ KB and $cwnd_{\text{cons}} = 80$ KB ($k = \lceil 230/80 \rceil = 3$ rounds, $B_{\text{budget}} = 3 \times 80 - 230 = 10$ KB), it is sufficient for approximately one P-frame per I-frame transmission, keeping base-layer frames flowing.

The conservative estimate for $cwnd_{\text{cons}}$ is critical. Using the raw bandwidth estimate would risk interleaving too aggressively, e.g., if the actual available bandwidth is lower than estimated, the interleaved P-frame packets would compete with I-frame packets for the same congestion window space, delaying the I-frame rather than helping the P-frame. The Kalman-filtered estimate provides a lower bound that is safe to fill without risking I-frame completion. If the budget computation yields zero or negative slack (meaning the I-frame exactly fills the available rounds), no interleaving occurs and the mechanism has no effect, it avoids making things worse.

Reconfiguration-aware scheduling targets findings F2–F4. Starlink employs synchronized reconfigurations at the 12th, 27th, 42nd, and 57th second of each minute, with jitter of $\sigma = 13.2$ ms and median duration of 58 ms (Section 4.2.4). Rather than building complex prediction models, we exploit this deterministic schedule directly. We define a *danger zone* of ±100 ms around each known reconfiguration time, and avoid scheduling I-frames on the Starlink path during these windows. The 200 ms total window is intentionally conservative, 92.2% of events fall within ±20 ms of their expected time, and 88%

complete within 100 ms. During a danger zone, I-frames are routed to the backup path, while P-frames continue on Starlink (they are small enough to tolerate occasional packet loss without catastrophic impact). Because four danger zones per minute occupy only 800 ms total (1.3% of time), the additional backup-path usage is minimal.

Dependency-aware co-location addresses SVC's inter-layer dependencies. Each enhancement-layer frame depends on the nearest preceding frame in a lower layer. If dependent frames are scheduled onto different paths with asymmetric RTTs (15 ms WiFi vs. 45 ms Starlink), the receiver must buffer the earlier-arriving frame until its dependency arrives on the slower path. The solution is to route each frame to the same path as the frame it depends on, using a lightweight scheduling history that records the path assignment for recently scheduled objects. Since dependency chains in SVC are short (at most 2–3 frames within a GOP), the history need only track one GOP's worth of entries (50 frames at 50 fps).

The co-location mechanism is conceptually simple but addresses a subtle failure mode. Without it, even a scheduler that correctly identifies the primary path for base-layer frames might route enhancement-layer frames to a different path under congestion pressure. The receiver would then see the enhancement frame arrive, for instance, 30 ms before its base-layer dependency (since the backup path has lower RTT), and would be forced to buffer it until the base frame arrives. For a 50 fps stream, 30 ms of buffering adds more than one frame period of latency to the display pipeline. By co-locating dependent frames, the receiver always sees them in dependency order with minimal inter-frame jitter.

Cost-sensitive path preference is to control costs. The MPQUIC results show that schedulers route the majority of traffic over WiFi (MinRTT: 90%, BLEST: 92%, Redundant: 100%) yet fail to meaningfully outperform single-path WiFi. This contradicts the Starlink-primary philosophy as users subscribe to flat-rate satellite service precisely to avoid metered cellular charges. The scheduling system encodes an explicit cost preference by labeling each path with a `cost_class` attribute (`free` for Starlink, `metered` for the backup) and defaulting to the cost-free path unless a higher-priority rule overrides it. Unlike the first three mechanisms, cost preference is a system constraint rather than a reaction to a measurement finding.

The cost-sensitive default also changes the semantics of secondary-path usage. In transport-only MPQUIC, the secondary path is used whenever the scheduler's local heuristic favors it, which turns out to be most of the time for latency-oriented schedulers like MinRTT. With cost-sensitive defaults, the secondary path is reserved for specific, high-value scenarios such as routing I-frames around reconfiguration danger zones. This transforms the secondary path from a general-purpose overflow path (which transport schedulers use wastefully) into a backup (which application-guided rules use selectively). The result is dramatically reduced secondary-path usage which aligns with the user's cost expectations while providing targeted protection for the traffic that benefits most.

4.5. MoMQ Rule Design for SVC

The four scheduling mechanisms identified above pose a bridging challenge that the SVC encoder knows about frame types, dependencies, and deadlines, while the network stack knows about path metrics, reconfiguration timing, and access cost. MoMQ bridges these two views through the declarative rule framework described in Chapter 3 that Objects carry metadata, paths carry labels, and rules match metadata to delivery preferences without requiring the relay to interpret media semantics. For the SVC-over-LEO use case, we define four rules that use `PRIORITY`, `BALANCING`, `PATH_PREFERENCE`, and `PATH_AFFINITY` actions. The subscriber maintains path labels via `PATH_LABEL_UPDATE`.

These four rules are matched in descending priority order, implementing the Rule Evaluation stage of the three-stage pipeline (Section 3.6). The highest-priority rule handles P-frame interleaving that when an I-frame is in flight (`pending_iframe=true`), arriving P-frames check the remaining interleave budget and are scheduled immediately within the current congestion-window round if they fit, or queued normally otherwise. The second rule handles reconfiguration avoidance that during danger zones (`leo_state=reconf`), I-frames are routed to the backup path; outside these windows the rule does not match and I-frames follow the cost-sensitive default. This is the only scenario in which I-frames deliberately traverse the metered path. The third rule enforces dependency co-location that enhancement-layer frames carrying a `depends_on` metadata field are routed to the same path as their

reference frame by consulting the scheduling history; if the dependency is not found, the frame falls through to lower-priority rules. The fourth and lowest-priority rule is the cost-sensitive default, routing all remaining frames to the `cost_class=free` path (Starlink). The hierarchical design ensures that backup-path usage remains limited to I-frames during reconfiguration danger zones.

The priority ordering matters because it determines which mechanism takes precedence when multiple rules would apply to the same Object. Consider an enhancement-layer P-frame that arrives during a danger zone while an I-frame is in flight. The interleaving rule (highest priority) checks the budget first; if the P-frame fits within the slack capacity, it is interleaved regardless of the danger zone. If it does not fit, the dependency co-location rule (third priority) routes it to the same path as its base-layer counterpart. The reconfiguration avoidance rule (second priority) applies only to I-frames, so it does not conflict with P-frame scheduling. This layered evaluation ensures that each mechanism handles its specific concern without blocking or overriding the others.

The subscriber maintains path labels that these rules consult through `PATH_LABEL_UPDATE` messages (Section 3.3). The *LEO Reconfiguration Marker* is tag-driven. During negotiation phase, subscriber labels this path with "starlink", which drives relay to add `leo_state=reconf` label to the Starlink path during ±100 ms danger windows. This method avoids communication latency. The *Interleave Budget Tracker* is event-driven at the relay, it computes B_{budget} when an I-frame begins transmission, deducts each interleaved P-frame's size, and resets when the I-frame completes. The *Scheduling History Tracker* records each Object-to-path assignment in a bounded hash map (one GOP's worth of entries) and provides lookups for the `PATH_AFFINITY` action. Controller overhead is negligible.

4.6. Simulated Ablation Study

Before deploying MoMQ on the live Starlink testbed, we use a controlled simulation to isolate the steady-state contributions of P-frame interleaving (Rule 1) and dependency co-location (Rule 3). Reconfiguration events are not modeled, so reconfiguration avoidance (Rule 2) is evaluated later in the live testbed, while the cost-sensitive default (Rule 4) remains fixed across all simulated configurations. Simulation serves two purposes here. First, it isolates these steady-state mechanisms under reproducible network conditions. Second, it provides the statistical power needed to distinguish configurations with confidence, running 250 runs per configuration.

The simulation models two asymmetric paths whose parameters are drawn from the testbed characterization in Section 4.2. The primary path has a base delay of 40 ms, delay jitter of 5 ms, packet-loss rate of 0.2%, and capacity of 80 Mbps, while the backup path has a base delay of 15 ms, delay jitter of 2 ms, packet-loss rate of 0.1%, and capacity of 50 Mbps. The SVC encoding parameters match the live setup (1080p, 50 fps, three temporal layers, 1-second GOPs), with LeoCC on the satellite path and Cubic on the backup path. Each of the 50 iterations executes five repetitions of four configurations in interleaved round-robin order, generating 250 runs per configuration. The four configurations therefore form a 2×2 factorial design around P-frame interleaving (Rule 1) and dependency co-location (Rule 3): the *baseline* keeps both mechanisms enabled, and three ablated variants disable co-location alone, interleaving alone, or both simultaneously. The cost-sensitive default (Rule 4) remains active in all configurations. The primary metric is the *p1–p99 jitter buffer*, the difference between the 99th and 1st percentiles of per-frame delivery deviation within each run, corresponding to the playback buffer needed to absorb 98% of frame-level timing variation. This metric is more robust to single-frame outliers than the min–max buffer used in the live evaluation, making it better suited for cross-run statistical comparison at scale.

Table 4.3 summarizes the results.

Table 4.3: Simulated ablation: playback buffer under incremental rule removal ($n = 250$ runs per configuration, 50 iterations × 5 repetitions). All pairwise comparisons significant at $p < 0.001$ (Mann–Whitney U).

Configuration	**Median (ms)**	σ **(ms)**	**IQR (ms)**	Δ%	**Cohen's** d
MoMQ baseline	237.4	37.5	45	—	—
w/o Rule 3 (co-location)	342.2	80.9	137	+44.3	1.77
w/o Rule 1 (interleaving)	395.9	72.6	88	+66.7	2.97
w/o Rules 1+3	501.5	116.3	181	+111.6	3.09

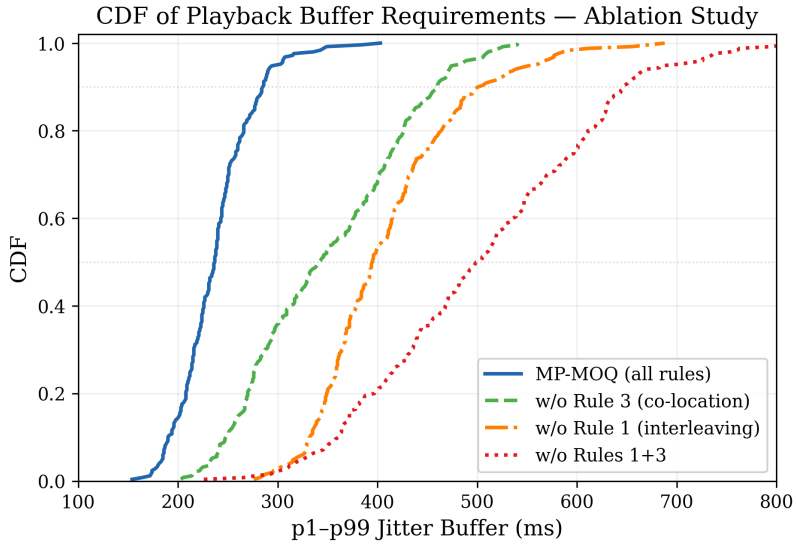

Figure 4.9: CDF of p1–p99 jitter buffer across 250 runs per configuration. The clear separation between curves confirms that both ablated mechanisms provide distinct, statistically significant contributions to buffer reduction.

With the baseline rule set, MoMQ achieves a median p1–p99 buffer of 237 ms (σ = 37.5 ms). Removing dependency co-location increases the buffer by 44.3% to 342 ms, as enhancement-layer frames split across paths with asymmetric RTTs force the receiver to buffer early-arriving frames until their dependencies arrive on the slower path. Removing P-frame interleaving is more damaging (+66.7%, median 395 ms), confirming that the convoy effect from multi-round I-frame transmission (Problem 1) dominates tail latency. Without interleaving, all P-frames in a GOP are blocked for the duration of the I-frame's 2–3 congestion-window rounds, inflating their completion time by 50–100 ms. Removing both mechanisms yields a 111.6% increase to 501.5 ms, almost exactly the sum of the individual degradations (44.3 + 66.7 = 111.0%). This near-additive behavior confirms that interleaving and co-location address independent problems with negligible interaction.

The CDF (Figure 4.9) reveals that the four configurations form clearly separated distributions. The 95th percentile of the baseline (303 ms) falls below the 5th percentile of the w/o-Rules 1+3 configuration (327 ms), meaning the two distributions overlap by less than 5%. Cohen's d values confirm large effect sizes ($d > 1.7$) for all baseline-vs.-ablated comparisons, and pairwise win rates reinforce the ordering: MoMQ outperforms the w/o-co-location variant in 90.0% of run-level comparisons, w/o-interleaving in 99.1%, and w/o-both in 99.3%. Even the comparison between removing co-location and removing interleaving yields a medium effect (d = 0.78), consistent with the 53 ms gap between their medians.

Figure 4.10 tracks the cumulative median-of-medians across iterations, demonstrating that the estimates converge by approximately iteration 30 and remain stable through iteration 50. The baseline converges fastest, reflecting its low variance (σ = 37.5 ms); the w/o-Rules 1+3 configuration, with the highest standard deviation (σ = 116.3 ms), requires approximately 35 iterations but achieves comparable stability thereafter. The four curves maintain consistent separation throughout, ruling out the possibility that the observed ordering is an artifact of finite sample size.

These results establish three findings that the live evaluation must confirm. First, P-frame interleaving contributes more to buffer reduction than dependency co-location (+66.7% vs. +44.3% degradation when removed). Second, the near-additive compounding of individual degradations (111.6% $\approx$ 44.3 + 66.7) confirms the design of the scheduling mechanisms. Third, the statistical significance of all pairwise comparisons ($p < 0.001$, Cohen's $d \geq 1.77$) and the convergence of estimates over 50 iterations establish that these performance differences reflect genuine effects of the rule-based scheduling design, not network variability. The next section evaluates whether these findings hold under the uncontrolled conditions of a live Starlink link.

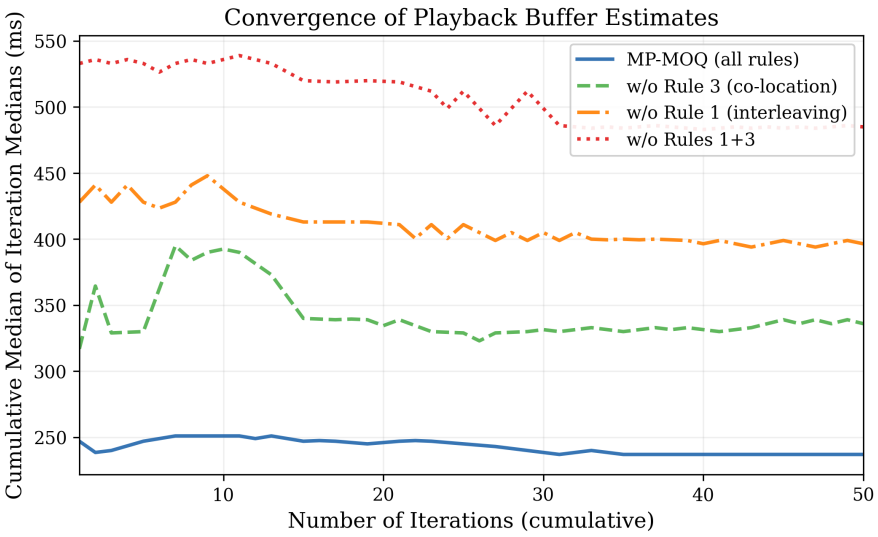

Figure 4.10: Convergence of cumulative median-of-medians estimates over 50 iterations. All configurations stabilize within ±5 ms of their final value by approximately iteration 30, confirming adequate statistical power.

4.7. Evaluation

On the live Starlink testbed, MoMQ is the only configuration that brings P99.9 below the 150 ms interactive target, reaching 114.1 ms with only 11.3% backup-path usage. Building on the controlled ablation in Section 4.6, the rest of this section unpacks that result in two steps: Table 4.4 first compares MoMQ against the single-path baselines and transport-only MPQUIC schedulers, and Table 4.5 plus Figure 4.11 then trace the gain back to the individual rules and frame-level mechanisms behind it.

Transport-only MPQUIC schedulers fail to improve upon single-path baselines at high percentiles. MinRTT's P99.9 (479.53 ms) is *worse* than single-path WiFi (399.85 ms) because I-frame overflow onto Starlink creates cross-path reordering and exposes critical traffic to reconfiguration events. Round-Robin is worse still (P99.9 694.88 ms) due to systematic per-round asymmetry penalties. BLEST and Redundant (P99.9 384.72–394.51 ms) largely match WiFi performance since both effectively operate as single-path WiFi for I-frame bursts and cannot prevent rare I-frame–reconfiguration collisions.

These results confirm that rule-based scheduling closes the tail-latency gap. MoMQ achieves a P99.9 of 114.1 ms, a 71% reduction from the best transport-only scheduler (BLEST, 384.72 ms), while routing only 11.3% of traffic to the backup path, compared to 90–100% for latency-oriented transport schedulers. The four rules together eliminate the tail events that transport-only schedulers cannot address, bringing P99.9 below the 150 ms interactive target for the first time across all configurations.

Incremental rule activation reveals complementary contributions (Table 4.5). The cost-sensitive default alone achieves a median buffer of 247.6 ms. Adding P-frame interleaving reduces the buffer to 198.4 ms by relieving head-of-line blocking during multi-round I-frame transmission. Adding reconfiguration avoidance provides the largest single improvement (151.9 ms, P99.9 dropping from 233.7 to 117.5 ms),

Table 4.4: Tail-latency and path-usage comparison across all configurations.

Configuration	**Median FCT (ms)**	**P99 (ms)**	**P99.9 (ms)**	**Backup Usage**
Copa / WiFi (single-path)	25.2	235.0	399.8	100%
LeoCC / Starlink (single-path)	32.6	258.4	541.7	0%
MPQUIC MinRTT	25.9	284.6	479.5	89.7%
MPQUIC Round-Robin	31.8	345.2	694.9	47.6%
MPQUIC BLEST	24.8	225.1	384.7	91.8%
MPQUIC Redundant	25.1	229.7	394.5	100% (dup.)
MoMQ (ours)	**28.2**	**85.6**	**114.1**	**11.3%**

Table 4.5: Ablation analysis: incremental rule activation.

Rule Set	Median Buffer (ms)	P99.9 FCT (ms)	Backup Usage
Cost-sensitive default only	247.6	277.3	0.3%
+ Interleaving	198.4	233.7	6.2%
+ Reconfiguration avoidance	151.9	117.5	9.4%
+ Dependency co-location (full)	127.9	114.1	11.3%

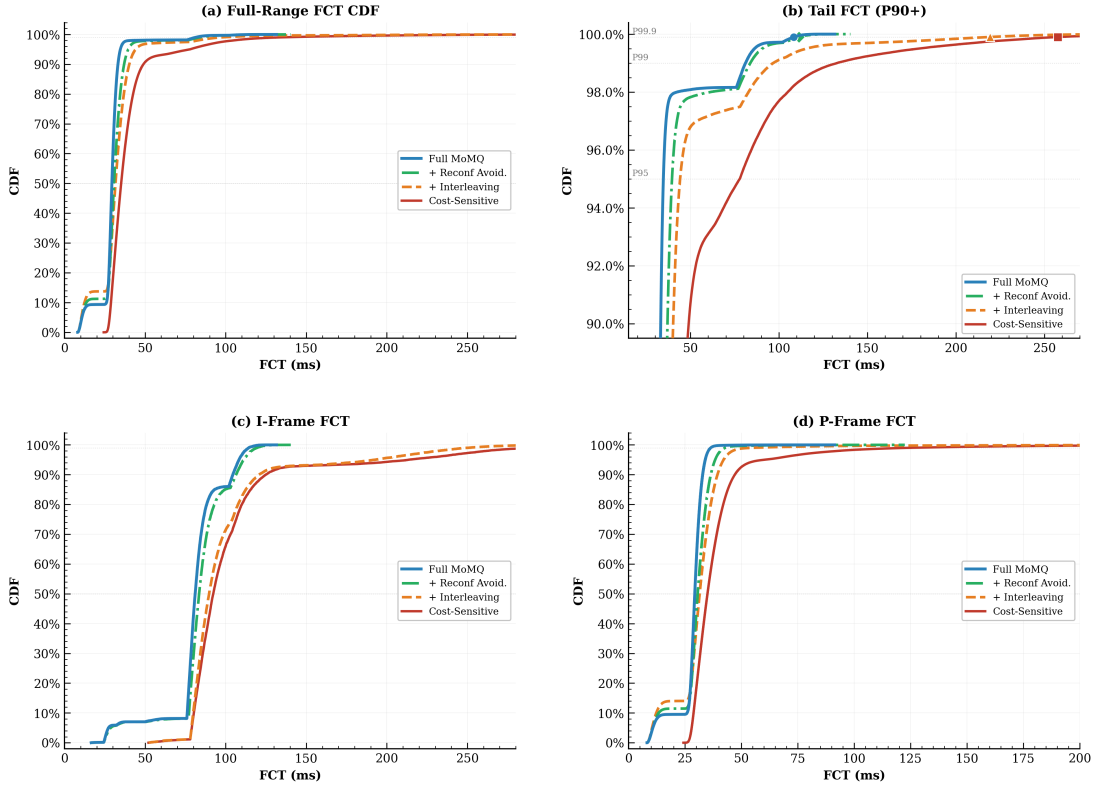

Figure 4.11: Frame completion time distributions under incremental rule activation (250 runs per configuration, ~750 000 frames per mode). (a) Full-range CDF. (b) Tail distribution (P90 and above). (c) I-frame FCT only. (d) P-frame FCT only.

as I-frame–reconfiguration collisions are proactively eliminated. Finally, dependency co-location brings the buffer to 127.9 ms by eliminating cross-path reordering of enhancement-layer frames. Each rule provides a distinct contribution which is consistent with the simulated ablation in Section 4.6.

To understand the mechanisms behind these incremental improvements, Figure 4.11 decomposes the per-frame completion time distribution across an extended measurement campaign of 50 iterations (250 runs per configuration, approximately 750 000 frames per mode). The four-panel layout separates the full-range FCT distribution (a), the tail above P90 (b), and individual I-frame (c) and P-frame (d) breakdowns, revealing which frame types each rule affects and why.

The full-range CDF in panel (a) exposes a structural signature of multipath scheduling. The three configurations that enable WiFi offloading exhibit a distinct step near 8–14 ms at the left end of the distribution, corresponding to P-frames routed via the WiFi path (one-way delay ≈ 8 ms). The cost-sensitive default, which confines all traffic to Starlink, produces no such step, its distribution onset begins near 25 ms, reflecting Starlink's one-way delay. Beyond 50 ms the tails diverge sharply. The tail analysis in panel (b) reveals the cause that at the P99.9 level, two distinct clusters emerge. The full system and +reconf-avoidance form a tight low-latency cluster near 108–112 ms, while cost-sensitive and +interleaving remain above 217 ms. The ~100 ms difference between these clusters quantifies the dominant contribution of reconfiguration avoidance. Without it, I-frame–reconfiguration collisions produce tail events that no amount of P-frame optimization can eliminate. This two-cluster pattern is

consistent with the P99.9 drop from 233.1 to 117.0 ms in Table 4.5 when reconfiguration avoidance is activated.

Panel (c) isolates I-frame behavior and pinpoints the mechanism. Cost-sensitive and +interleaving I-frames have very similar distributions (median 92 vs. 90 ms), confirming that P-frame interleaving does not affect I-frame scheduling, it operates within the slack capacity of I-frame congestion-window rounds without altering the I-frame's own path assignment (Section 4.4). The separation begins with reconfiguration avoidance: rerouting I-frames to WiFi during danger windows produces a visible low-FCT step in the 16–50 ms range, corresponding to the 6.7% of I-frames that traverse WiFi during the four ±100 ms danger windows per minute. This selective rerouting compresses I-frame P99 from 286 ms to 119 ms (−58%), eliminating the cwnd-collapse penalty that satellite reconfiguration collisions impose (Problem 2). Full MoMQ adds a further modest reduction (I-frame P99: 119 → 114 ms) through dependency co-location, which anchors enhancement-layer frames to the same path as their I-frame reference and reduces cross-path scheduling interference.

Panel (d) shows that P-frame tail compression is the main reason for the buffer improvements in Table 4.5. Under cost-sensitive scheduling, P-frames have high tail latency, with a P99/P50 ratio of 3.5× (P99 = 122 ms, P50 = 35 ms). This is mainly driven by head-of-line blocking while multi-round I-frames are being transmitted (Problem 1). Each I-frame occupying the congestion window for 2–3 RTTs delays all queued P-frames by 50–100 ms, inflating the P-frame tail far beyond its intrinsic one-way delay. Interleaving offers the most targeted relief, e.g., by filling the slack capacity within I-frame transmission rounds with P-frame packets, it reduces P-frame P99 from 122 ms to 54 ms (−56%). Reconfiguration avoidance compresses P-frame P99 further to 43 ms by eliminating the cascading effect of I-frame–reconfiguration collisions on subsequent P-frames. The full system is highly stable, with a P99/P50 ratio of just 1.2× (P99 = 36 ms, P50 = 29 ms). Almost all P-frames complete within a single RTT of their intrinsic one-way delay. This 70% reduction in P-frame P99 explains why playback buffers shrink under MoMQ: stalls are triggered by clusters of late-arriving P-frames, and the convoy effect (Problem 1) is precisely the source of such clusters.

4.8. Location Sensitivity

The evaluation so far uses a relay in Nuremberg, Germany, roughly 700 km from the subscriber in the Netherlands. Whether the results depend on this particular subscriber–relay topology is an open question. We test two alternative configurations at longer distances.

4.8.1. Canada Subscriber

We move the subscriber to British Columbia, Canada, placing it roughly 6 000 km from the relay, which stays in Germany. The subscriber connects to the relay via both Starlink and a terrestrial WiFi path as before; the publisher remains in the Netherlands. This subsection focuses on the multipath comparison between MoMQ with all four rules against MoQ over MPQUIC and the BLEST scheduler.

Table 4.6 summarizes the results. MoMQ achieves median FCT of 136.2 ms versus 143.4 ms for BLEST (5.0% better) and buffer medians of 425.3 ms versus 455.3 ms (6.6% better). These improvements are far smaller than with the Germany relay, where MoMQ reduced P99.9 FCT by roughly 70% and buffer by about 60%.

The CDF curves (Figure 4.12) confirm this. FCT distributions in panel (a) overlap heavily between 100 and 150 ms, with MoMQ ahead only in the tail above 170 ms. Panel (b) shows a clearer but still modest buffer separation: MoMQ's distribution shifts left, yet both methods require buffers above 350 ms.

Table 4.6: Multipath comparison with subscriber in British Columbia, Canada.

Metric	**MoMQ**	**MoQ BLEST**	**Diff**
FCT Median (ms)	136.2	143.4	5.0%
FCT Std (ms)	26.4	49.0	—
Buffer Median (ms)	425.3	455.3	6.6%
Buffer Std (ms)	27.1	32.5	—

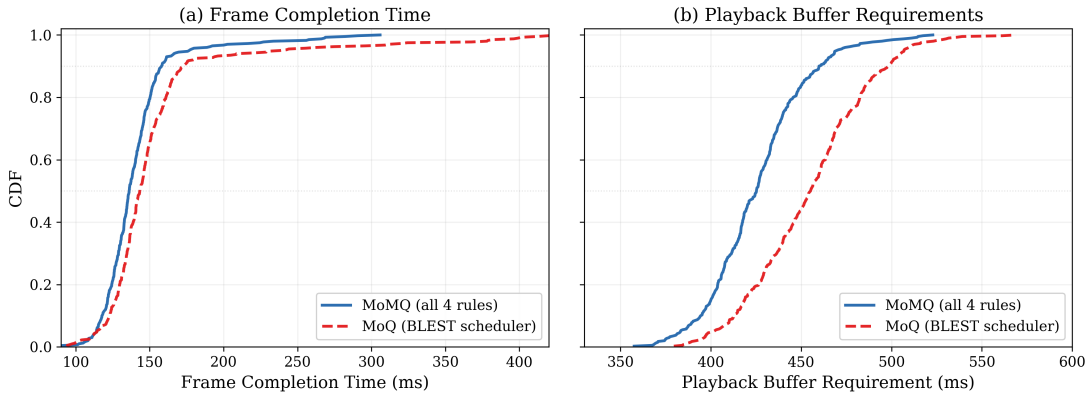

Figure 4.12: MoMQ vs. MoQ/BLEST with subscriber in Canada. (a) FCT CDF. (b) Buffer CDF.

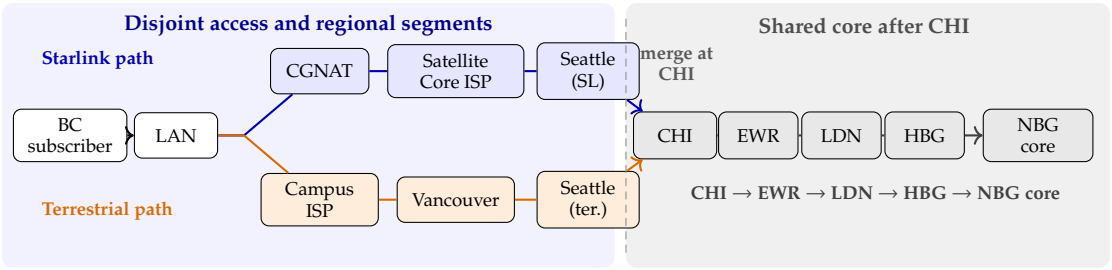

Figure 4.13: Schematic traceroute-derived path sketch for the Canada subscriber experiment. The first unambiguous shared router appears at Chicago; Table 4.7 lists the matched hops from the raw logs.

Table 4.7: Traceroute evidence for path convergence in the Canada subscriber experiment.

Evidence point	Raw-hop evidence	Interpretation
Seattle region	**Starlink:** hop 6 reaches `sea-b1-link` (213.155.141.32). **Terrestrial:** hops 12–13 reach `sea-b4-link` and `sea-b1-link` (62.115.138.40, 62.115.132.157).	Same metro, but different interfaces; Seattle alone does not prove convergence.
First identical router	**Starlink:** hop 7 reaches `chi-bb2-link` (62.115.132.154). **Terrestrial:** hop 14 reaches the same router and address.	First unambiguous convergence point.
Shared Twelve99 backbone	**Starlink:** hops 8–11 traverse `ewr-bb2`, `ldn-bb2`, `hbg-bb2`, and `hbg-b2`. **Terrestrial:** hops 15–18 traverse the same router sequence.	Identical hostnames and IPs from CHI through HBG; the `bb2` hops also expose MPLS label 415780 on both traces.
Hetzner ingress	**Starlink:** hops 12–14 traverse `hetzner-ic-383976`, `core22.fsn1`, and `core11.nbg1`. **Terrestrial:** hops 19–21 traverse `hetzner-ic-383975`, `core21.fsn1`, and `core11.nbg1`.	Adjacent Hetzner-facing interfaces and core nodes, but not a fully identical suffix yet.
Destination-side core	**Starlink:** hop 15 reaches `core-spine-rdev2` (213.239.239.122); hops 18–19 reach the same host. **Terrestrial:** hop 22 reaches the same core spine; hops 25–26 reach the same host.	Paths rejoin inside Hetzner before the target server.

The limited improvement is explained by the path convergence shown in Figure 4.13 and aligned to the raw traceroute hops in Table 4.7. Both traces enter Twelve99 in the Pacific Northwest, but Seattle alone does not prove convergence: the Starlink trace reaches `sea-b1-link` at hop 6 (213.155.141.32), whereas the terrestrial trace reaches `sea-b4-link`/`sea-b1-link` at hops 12–13 on different interface addresses. The first unambiguous common router is `chi-bb2-link.ip.twelve99.net` (62.115.132.154), appearing at hop 7 on Starlink and hop 14 on the terrestrial trace. From there the paths follow the same Twelve99 backbone sequence through Newark, London, and Hamburg; the repeated MPLS label 415780 on those backbone hops further supports a shared core segment. The traces then pass through adjacent Hetzner-facing interfaces and rejoin at `core-spine-rdev2.cloud1.nbg1.hetzner.com` before the target host. Once traffic enters this shared long-haul segment, both paths experience nearly the same queueing and propagation conditions, leaving multipath scheduling with limited leverage. Reconfiguration avoidance contributes little because the shared core tail masks Starlink reconfiguration events, and P-frame interleaving gains nothing when both paths deliver frames at comparable latencies. MoMQ's scheduling rules assume path diversity that does not exist on this topology.

Table 4.8: Baseline comparison with relay in Helsinki, Finland.

Configuration	PF Med (ms)	IF Med (ms)	All P99.9 (ms)	Buffer (ms)	Backup
Copa / WiFi	34.3	142.3	364.5	396.6	100.0%
LeoCC / Starlink	53.5	186.6	450.2	440.1	0.0%
MPQUIC BLEST	33.7	132.9	275.3	371.5	84.3%
MoMQ (ours)	**45.1**	**95.9**	**132.4**	**164.3**	**9.3%**

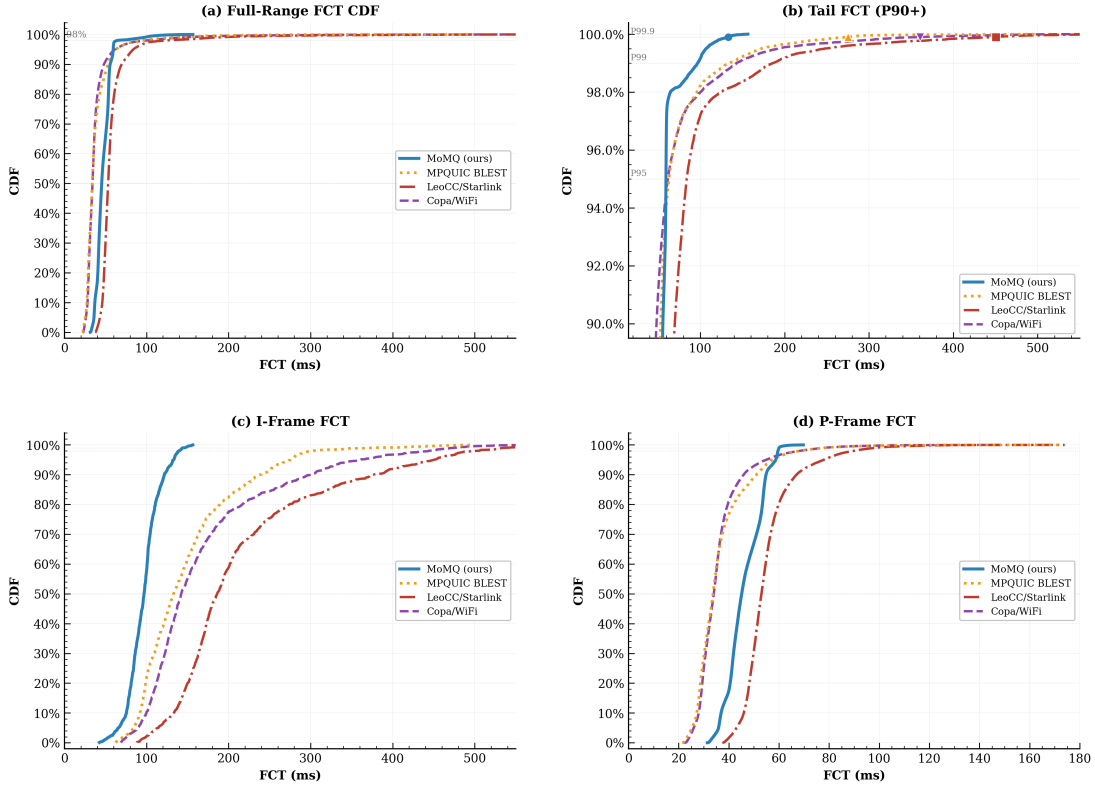

Figure 4.14: FCT distributions with Finland relay. (a) Full-range CDF. (b) Tail (P90+). (c) I-frame FCT. (d) P-frame FCT. The same structural patterns from the Germany evaluation hold at higher relay distance.

4.8.2. Finland Relay

The Canada result suggests that MoMQ's gains require genuine path diversity, not just nominally separate network interfaces. To test this hypothesis, we move the relay to a Hetzner datacenter in Helsinki, Finland, roughly 1 800 km from the subscriber. Before running the experiment, we used traceroute to verify that the WiFi and Starlink paths from the Netherlands to Finland do not share routers. The two paths remain distinct through separate transit networks all the way to the relay. One-way delays are approximately 30 ms via WiFi and 50 ms via Starlink. Video parameters, rule configuration, and measurement methodology match Section 4.2.1, except that each configuration runs for 120 seconds instead of 100 seconds. We run 10 repetitions per configuration.

Table 4.8 compares the same four configurations tested in Section 4.7. The baselines produce P99.9 values between 275.3 and 450.2 ms and need playback buffers of 371.5–440.1 ms, while MoMQ keeps P99.9 at 132.4 ms and buffer at 164.3 ms, corresponding to reductions of 52–70%.

The FCT distributions (Figure 4.14) shows similar results. Panel (a) preserves the 98% step from P-frames completing quickly, with the last 2% of I-frames stretching the tail. Panel (c) shows MoMQ finishing nearly all I-frames below 150 ms, while all three baselines extend to 400–550 ms. The buffer distributions (Figure 4.15) confirm the gap: MoMQ requires 164.3 ms against 371–440 ms for all baselines.

Table 4.9 breaks down the rule contributions. Adding interleaving cuts the buffer from 289.0 ms to

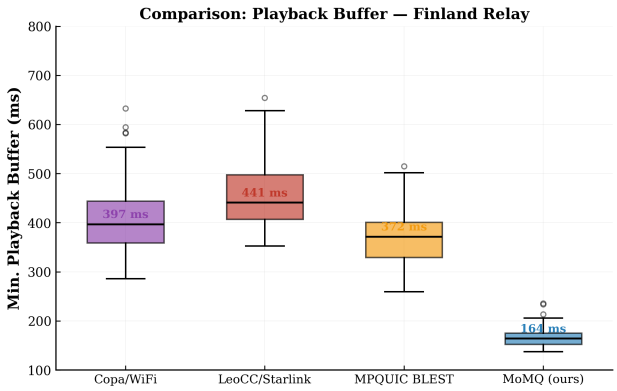

Figure 4.15: Playback buffer requirements with Finland relay.

Table 4.9: Incremental rule activation with Finland relay.

Rule Set	PF Med (ms)	IF Med (ms)	P99.9 (ms)	Buffer (ms)	Backup
Cost-sensitive default	68.0	109.4	309.3	284.0	0.9%
+ Interleaving	66.3	107.2	232.8	234.7	4.8%
+ Reconf. avoidance	56.4	99.8	133.5	175.3	7.1%
Full MoMQ	45.1	95.9	132.4	164.3	9.3%

234.7 ms. Reconfiguration avoidance brings it to 175.3 ms, and the full rule set reaches 164.3 ms. P99.9 FCT drops from 309.3 ms to 132.4 ms, a 57% reduction.

Each rule contributes meaningfully. Interleaving alone cuts P99.9 from 309.3 ms to 232.8 ms (25% reduction) and the buffer from 284.0 ms to 234.7 ms. Reconfiguration avoidance adds a further reduction to 175.3 ms, and the full rule set reaches 164.3 ms. Together these two mechanisms account for nearly all of the P99.9 improvement.

The ablation CDF (Figure 4.16) visualizes this progression. Panel (a) shows the four configurations forming separated distributions, with reconfiguration avoidance producing the largest single shift leftward. Panel (b) confirms that Full MoMQ and +reconf. avoidance nearly overlap in the tail above P95, while Cost-Sensitive and +Interleaving extend past 200 ms. In panel (c), Full MoMQ finishes 90% of I-frames below 110 ms, while Cost-Sensitive needs up to 130 ms. Panel (d) reveals a clear P-frame separation between the two modes that include reconfiguration avoidance and the two that do not.

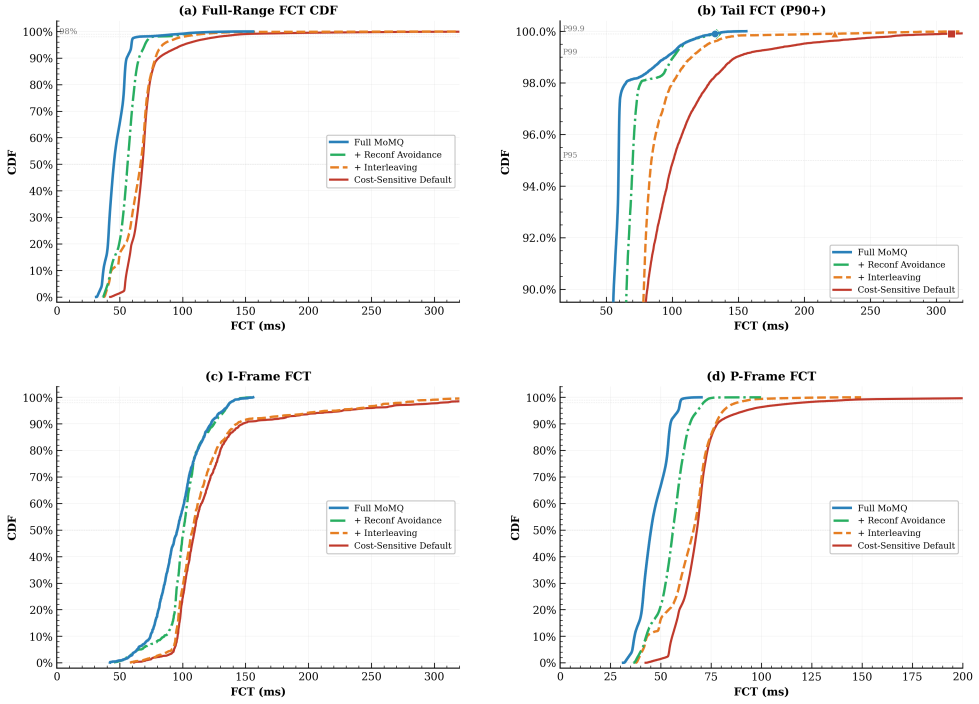

Figure 4.16: FCT distributions for the Finland relay ablation. (a) Full-range CDF. (b) Tail (P90+). (c) I-frame FCT. (d) P-frame FCT.

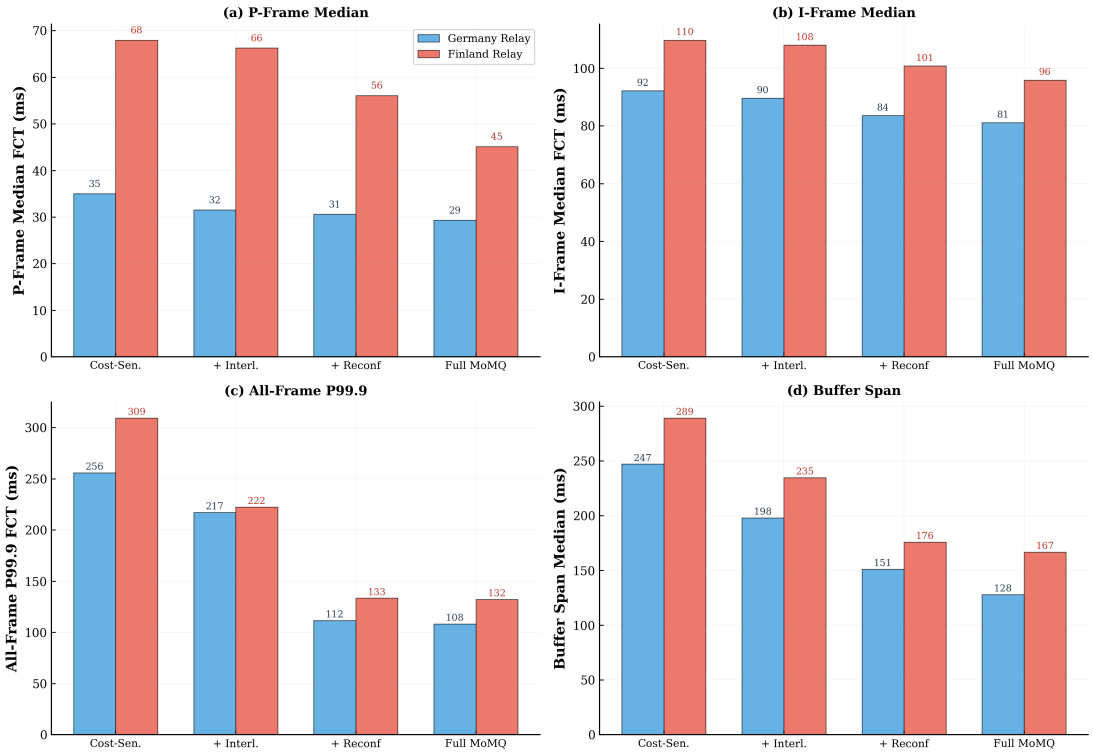

Figure 4.17: Germany vs. Finland relay: per-metric comparison across ablation modes. (a) P-frame median FCT. (b) I-frame median FCT. (c) All-frame P99.9 FCT. (d) Buffer span median. P-frame medians increase most (55–94%) due to the proportionally larger effect of added propagation delay on small frames; other metrics rise by 2–30%. The relative ordering stays the same.

4.8.3. Cross-Location Comparison

Figure 4.17 compares the Germany and Finland results side by side across all four ablation modes. Moving the relay to Finland increases all metrics, though not uniformly. P-frame medians rise by 55–94% because the added propagation delay has a larger relative effect on these small, fast frames. I-frame medians increase by 19–20%, P99.9 values by 2–22%, and buffer requirements by 17–30%. Full MoMQ outperforms all other configurations in both locations.

The MoMQ rules from Section 4.5 were not tuned for any specific relay location. They use metadata fields and path labels, not absolute latency thresholds. The same rules work at 2.5× longer relay distance with unchanged ranking. Yet on a topology where path diversity does not exist (Canada), the gains nearly disappear. This confirms the design premise that genuine multipath diversity drives MoMQ's scheduling benefits, and the transport-agnostic rule framework leverages that diversity regardless of absolute distance.

4.9. Summary

This chapter has demonstrated that for SVC video over heterogeneous LEO-plus-terrestrial paths, the limiting issue is not raw multipath availability but semantic mismatch [11]. Transport-only schedulers cannot simultaneously encode frame criticality, dependency order, reconfiguration awareness, and cost preference, for example, MinRTT and BLEST push 90–92% of traffic to the metered path without improving tail latency; Round-Robin introduces systematic reordering that worsens performance; and Redundant doubles bandwidth for negligible gain. MoMQ resolves this by adding a narrow declarative control plane atop MoQT relay forwarding [30]. Four rules, namely P-frame interleaving, reconfiguration avoidance, dependency co-location, and cost-sensitive default collectively reduce tail FCT while keeping backup-path usage minimal. The simulated ablation study (Section 4.6) confirms that these contributions are statistically robust across 250 independent trials, with all pairwise comparisons significant at $p < 0.001$ and effect sizes exceeding $d = 1.7$, the live evaluation (Section 4.7) validates

them under uncontrolled network conditions, and the location sensitivity study (Section 4.8) shows that these gains require genuine path diversity—they nearly vanish when paths converge (Canada) but hold at a longer distance when paths remain distinct (Finland).

The measurement study contributes independently of MoMQ. The characterization of Starlink reconfiguration timing (σ = 13.2 ms, 92.2% within ±20 ms of predicted time) and duration (median 58 ms, 88% within 100 ms) provides a reference for any protocol designer working with LEO satellite links. The identification of four distinct scheduling problems, namely head-of-line blocking, completion-time inflation, dependency-order violations, and reactive overuse, creates a diagnostic framework applicable beyond SVC that any application with heterogeneous data importance, inter-object dependencies, and cost-asymmetric paths will encounter some subset of these problems.

The mechanism design also illustrates a general pattern for applying MoMQ to new domains. The process follows four steps: measure to identify problems, trace each problem to a missing piece of application context, design a mechanism that uses that context, and express the mechanism as a declarative rule. This pattern could be replicated for cloud gaming (where input-response frames have different latency requirements than background rendering), live sports broadcasting (where camera switches create transient priority changes), or augmented reality (where pose-update Objects must arrive before the next rendering deadline). In each case, the application provides the domain knowledge through metadata and rules, and MoMQ provides the scheduling infrastructure.

The measurement study, mechanism design, rule specification, and evaluation together validate the central claim of this thesis that metadata-driven scheduling through MoMQ can close the tail-latency gap that transport-only multipath cannot.

5. Conclusion

This thesis studies real-time media delivery over multipath networks with application-aware scheduling. The central observation is that transport-layer multipath schedulers and application-layer media protocols operate with different notions of what matters. The transport scheduler optimizes for bytes, packets, and path metrics, while the application cares about frames, dependencies, and deadlines. This thesis bridges MoQT's Object model [30] and MPQUIC path scheduling [12] through a minimal extension that preserves relay scalability and demonstrates its application to SVC video conferencing over heterogeneous terrestrial-LEO paths.

5.1. Summary of Contributions

This thesis makes three main contributions, each aimed at closing the semantic gap between media applications and transport scheduling without complicating the relay architecture.

5.1.1. MoMQ Protocol Design

The first contribution is MoMQ itself, one setup parameter (`ENABLE_MOMQ`) and four control messages (`PATH_STATE_REPORT`, `PATH_LABEL_UPDATE`, `PATH_MAPPING_RULE`, `PATH_MAPPING_RESULT`) that allow metadata-driven delivery guidance. The design keeps relays application-agnostic, a relay evaluating a rule that matches `frame_type=IDR` does not need to know what an IDR frame is or why it matters. It expresses policies as preferences over path labels rather than path bindings, so rules remain valid when the network topology changes. It stays minimal, only five new wire elements total, to reduce implementation burden and facilitate incremental deployment. And it retains advisory semantics so relay policy and resource limits remain authoritative. The application's scheduling preferences are hints, not commands.

The four design principles, namely application-agnostic relays, transport-agnostic rules, minimal extension, and advisory semantics, are not arbitrary constraints. Each one addresses a specific deployment concern. Relays must serve diverse applications without per-application logic; rules must survive network changes without reissuing; the extension must be small enough that existing MoQT implementations can adopt it incrementally; and relays must retain autonomy to enforce fairness and resource limits.

5.1.2. SVC Conferencing Analysis and Strategy

The second contribution is a frame-aware scheduling strategy for SVC conferencing, grounded in measurement. It analyses four scheduling problems under transport-only behavior, namely IDR head-of-line blocking, IDR completion-time inflation, dependency-order violations, and reactive secondary-path overuse, and traces each one to a specific mismatch between what the transport layer sees (bytes and packets) and what the application needs (frame-level scheduling). The measurement study shows that these problems are not hypothetical, they appear consistently in real Starlink measurements, producing tail latencies that exceed the 150 ms interactive budget by factors of 3–5×.

Each problem maps to a concrete scheduling mechanism: P-frame interleaving for head-of-line blocking, reconfiguration-aware scheduling for completion-time inflation, dependency-aware co-location for order violations, and cost-sensitive default routing for secondary-path overuse. These mechanisms are then expressed as declarative MoMQ rules that the relay can evaluate without understanding SVC encoding. The rule set demonstrates that the four action types defined by MoMQ (priority, balancing, path preference, path affinity) are sufficient to express non-trivial media scheduling policies.

5.1.3. Hybrid Terrestrial–LEO Integration

The third contribution is integration with heterogeneous terrestrial and LEO paths. LEO satellite networks present a challenging combination of high capacity, variable latency, periodic disruptions, and complementary failure modes relative to terrestrial links. The study shows that these characteristics are not merely obstacles but schedulable resources, namely the deterministic reconfiguration schedule of Starlink can be predicted and avoided, the high capacity of the satellite path can absorb bulk traffic at flat-rate pricing, and the low latency of the terrestrial path can serve time-critical frames.

The key finding is that transport-agnostic rules can exploit this complementarity without topology-specific configuration. The same rules that request "prefer the path labeled cost_class=free" work regardless of whether the free path is satellite, fiber, or WiFi—the abstraction layer provided by path labels decouples the scheduling intent from the physical network. The location sensitivity study (Section 4.8) nevertheless reveals an important boundary condition that these gains depend on genuine path diversity. When the subscriber is moved intercontinentally to Canada, placing it roughly 6 000 km from the relay, traceroute suggests the two paths converge onto shared transatlantic backbone routers, and MoMQ's advantage over BLEST shrinks to a 5% FCT difference. Moving the relay instead to Finland (1 800 km from the subscriber), where the paths remain distinct, reproduces the Germany results with higher absolute latencies and the same relative ordering across all configurations. The lesson is that MoMQ does not create path diversity—it exploits whatever diversity the topology provides.

5.2. Limitations

5.2.1. Prototype Scope

The evaluation is conducted on a controlled testbed with a specific hardware configuration (Starlink terminal, campus WiFi), a single satellite constellation (Starlink), and a single SVC operating point (1080p, 50 fps, three temporal layers, 1-second GOPs). These choices make the measurements interpretable, but they also bound the claims. Different SVC configurations could change frame-size asymmetry, dependency depth, and the relative value of interleaving or co-location, while other constellations may exhibit different handover periodicity, gateway placement, inter-satellite-link usage, and pricing models, changing the usefulness of reconfiguration-aware and cost-sensitive rules. The location sensitivity study (Section 4.8) shows that results generalize when genuine path diversity exists, but also that intercontinental topologies where paths converge limit the achievable gains. Production deployments would involve additional complexity, such as multiple concurrent subscribers, competing traffic from other applications, relay load balancing, and operational monitoring. Evaluating MoMQ under these conditions requires a production-grade implementation beyond the scope of this thesis.

For reproducibility and code navigation, Appendix A summarizes the structure of the companion `moq-examples` repository used for the prototype implementation and experiment tooling.

5.2.2. Rule Design Complexity

The quality of MoMQ's scheduling depends entirely on the quality of the rules installed by the subscriber. Well-designed rules can exploit application semantics to improve delivery; poorly designed rules, such as overly broad matches, conflicting actions, or incorrect metadata assumptions, can produce scheduling behavior that is worse than the transport-layer default. The current design provides no mechanism for validating rule correctness beyond syntactic checks, the relay accepts or rejects rules based on format, not on whether they will produce useful scheduling behavior.

5.2.3. Relay Trust Model

MoMQ assumes cooperative relays that faithfully evaluate rules and report path state accurately. In practice, relay operators may have incentives to deviate, for example, throttling certain traffic patterns, misreporting path availability, or ignoring resource-intensive rules. Advisory semantics accommodate benign deviation (relays overriding rules under resource pressure), but the protocol does not detect or prevent malicious deviation. The metadata exchanged between subscriber and relay, including object properties, path labels, scheduling preferences, also creates privacy exposure, e.g., a relay can infer the subscriber's encoding format, traffic pattern, and cost preferences from the rules it receives.

5.3. Future Work

5.3.1. Automated Rule Generation

Manual rule design works for well-understood scenarios like SVC conferencing, but it does not scale to diverse applications or dynamically changing network conditions. The rule set derived in Chapter 4 required understanding both SVC's encoding structure and the Starlink path's reconfiguration behavior—knowledge that took careful measurement to obtain and that may not transfer directly to other encoding formats or constellations.

The most practical next step is not to search the full MoMQ rule language directly, but to separate rule structure from parameter tuning. Chapter 4 already provides a compact template library grounded in observed failure modes: interleaving for multi-round I-frame blocking, danger-window avoidance for reconfiguration collisions, path affinity for dependency preservation, and cost-sensitive defaults for access asymmetry. An automated system could treat these as candidate rule families and infer when each one is warranted from relay-side traces that already exist in the evaluation setting, namely Object metadata, chosen path, completion time, reorder delay, rebuffer events, and backup-path usage. For example, repeated tail events concentrated on objects with `frame_type=IDR` during intervals labeled `leo_state=reconf` would justify a reconfiguration-avoidance rule, while persistent reorder delay for objects carrying `depends_on` would justify path affinity. This keeps the search space interpretable and ties learned policies to measurable phenomena rather than opaque controller behavior.

Within such a template set, online parameter adaptation is the most immediately feasible direction. The current SVC rule set already contains tunable quantities, such as the width of the danger window around predicted reconfigurations, the conservative interleaving budget derived from filtered bandwidth and minimum RTT, the thresholds that determine when traffic is treated as latency-critical, and policy caps on backup-path usage. These parameters can be updated from telemetry rather than fixed a priori. If observed reconfiguration durations drift upward, the danger window can widen automatically; if interleaving no longer reduces P-frame tail latency, the budget can shrink or be disabled; if backup-path usage exceeds a cost target, override conditions can tighten. Because these are scalar or low-dimensional adjustments, they are far more tractable than learning arbitrary rule programs and fit naturally within adaptive-control or bandit-style tuning loops [25].

More ambitious methods could learn limited rule structure from experience, but only under strong constraints. Reinforcement learning over the full MoMQ rule language is possible in principle, yet the search space is combinatorial (match conditions $\times$ action types $\times$ parameters), the reward is driven by rare tail events, and unsafe exploration would directly degrade media quality. A more realistic intermediate step is constrained structure learning, for example, compose policies from a small library of match predicates (such as `frame_type`, `temporal_layer`, `depends_on`, and path labels), evaluate candidate rules offline on recorded traces, and deploy only those that outperform the current policy under explicit cost and stability constraints. Cross-application transfer remains open. SVC conferencing, cloud gaming, and live streaming all contain objects with unequal criticality, but the useful predicates, acceptable trade-offs, and feedback signals will differ by domain. For that reason, automated rule generation is likely the most important next step for turning MoMQ from a hand-tuned prototype into a broadly deployable scheduling framework.

5.4. Concluding Remarks

Real-time media increasingly runs over heterogeneous paths. A user's device may be connected through WiFi, cellular, and satellite simultaneously, each link offering different capacity, latency, cost, and reliability. Multipath transport can exploit this diversity, but practical gains require scheduling decisions that reflect content semantics, namely which frames matter, which can tolerate delay, which should travel together, and which paths are appropriate for which traffic.

MoMQ provides this link through a small, layered extension. Applications express intent through metadata-driven rules, and relays map that intent to runtime path decisions without understanding the application's media format. The design is deliberately narrow because protocol extensions must be simple enough to deploy incrementally and robust enough to survive the diversity of real-world

networks. For SVC conferencing over terrestrial-LEO paths, this approach turns what would be uncontrolled scheduling into targeted, predictable media delivery. The strongest evidence appears in the live Starlink evaluation of Chapter 4, where the full rule set reduces P99.9 frame completion time to 114.1 ms while keeping backup-path usage at 11.3%. That is the moment where the thesis stops being a protocol-design argument and becomes a measured result on a real terrestrial–LEO path. A relay given a small amount of frame-level semantics can keep interactive media within budget instead of merely balancing bytes across links.

6. Declaration of Generative AI Use

This chapter documents the use of generative AI tools in the research and writing of this thesis.

I have used Claude Opus 4.6 in Github Copilot for debugging and refactoring code, as well as for generating experiment scripts and test cases. I have also used it for code suggestions while writing the implementation.

For writing, I have used GPT 5.4 to optimize my phrasing and check grammar. I have thoroughly reviewed the outcomes and manually edited the generated text word by word to ensure it actually reflects my intended meaning.

I here declare my use of generative AI tools does not violate academic integrity.

References

[1] Harald Alvestrand. *Overview: Real-Time Protocols for Browser-Based Applications*. RFC 8825. Internet Engineering Task Force, Jan. 2021. DOI: 10.17487/RFC8825.

[2] Venkat Arun and Hari Balakrishnan. "Copa: Practical Delay-Based Congestion Control for the Internet". In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*. Renton, WA: USENIX Association, 2018, pp. 329–342.

[3] Abdelhak Bentaleb et al. "A Survey on Bitrate Adaptation Schemes for Streaming Media Over HTTP". In: *IEEE Communications Surveys & Tutorials* 21.1 (2019), pp. 562–585. DOI: 10.1109/COMST.2018.2862620.

[4] Debopam Bhattacherjee and Ankit Singla. "Network Topology Design at 27,000 km/hour". In: *Proceedings of the 15th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '19)*. ACM, 2019, pp. 341–354. DOI: 10.1145/3359989.3365407.

[5] Mike Bishop. *HTTP/3*. RFC 9114. Internet Engineering Task Force, June 2022. DOI: 10.17487/RFC9114.

[6] Steven Blake et al. *An Architecture for Differentiated Services*. RFC 2475. Internet Engineering Task Force, Dec. 1998. DOI: 10.17487/RFC2475.

[7] Neal Cardwell et al. "BBR: Congestion-Based Congestion Control". In: *ACM Queue* 14.5 (2016), pp. 20–53.

[8] Neal Cardwell et al. *BBRv3: Algorithm Bug Fixes and Public Internet Deployment*. IETF 117, Presentation to CCWG. 2023.

[9] Gaetano Carlucci et al. "Analysis and Design of the Google Congestion Control for Web Real-Time Communication (WebRTC)". In: *Proceedings of the 7th ACM International Conference on Multimedia Systems (MMSys '16)*. ACM, 2016, 13:1–13:12. DOI: 10.1145/2910017.2910605.

[10] Cisco Systems. *Cisco Annual Internet Report (2018–2023)*. White Paper. 2020.

[11] David D. Clark and David L. Tennenhouse. "Architectural Considerations for a New Generation of Protocols". In: *Proceedings of the ACM SIGCOMM 1990 Conference*. ACM, 1990, pp. 200–208. DOI: 10.1145/99508.99553.

[12] Quentin De Coninck et al. *Multipath Extension for QUIC*. Internet-Draft draft-ietf-quic-multipath-17. Work in Progress. Internet Engineering Task Force, 2025.

[13] Mo Dong et al. "PCC Vivace: Online-Learning Congestion Control". In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*. USENIX Association, 2018, pp. 343–356.

[14] Simone Ferlin et al. "BLEST: Blocking Estimation-based MPTCP Scheduler for Heterogeneous Networks". In: *2016 IFIP Networking Conference (Networking) and Workshops*. IEEE, 2016, pp. 431–439.

[15] Alan Ford et al. *TCP Extensions for Multipath Operation with Multiple Addresses*. RFC 8684. Internet Engineering Task Force, Mar. 2020. DOI: 10.17487/RFC8684.

[16] Sadjad Fouladi et al. "Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol". In: *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18)*. USENIX Association, 2018, pp. 267–282.

[17] Sangtae Ha, Injong Rhee, and Lisong Xu. "CUBIC: A New TCP-Friendly High-Speed TCP Variant". In: *ACM SIGOPS Operating Systems Review*. Vol. 42. 5. 2008, pp. 64–74.

[18] Mark Handley. "Delay is Not an Option: Low Latency Routing in Space". In: *17th ACM Workshop on Hot Topics in Networks (HotNets '18)*. ACM, 2018, pp. 85–91. DOI: 10.1145/3286062.3286075.

[19] ITU-T. *One-Way Transmission Time*. ITU-T Recommendation G.114. May 2003.

[20] Jana Iyengar and Martin Thomson. *QUIC: A UDP-Based Multiplexed and Secure Transport*. RFC 9000. Internet Engineering Task Force, May 2021. DOI: 10.17487/RFC9000.

[21] Liz Izhikevich et al. “Democratizing LEO Satellite Network Measurement”. In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)* 8.1 (Feb. 2024). DOI: 10.1145/3639039.

[22] Van Jacobson. “Congestion Avoidance and Control”. In: *Proceedings of the ACM SIGCOMM 1988 Conference*. ACM, 1988, pp. 314–329. DOI: 10.1145/52324.52356.

[23] Ziyang Lai et al. “LeoCC: Making Internet Congestion Control Robust to LEO Satellite Dynamics”. In: *Proceedings of the ACM SIGCOMM 2025 Conference*. ACM, 2025. DOI: 10.1145/3718958.3750491.

[24] Yeon-sup Lim et al. “ECF: An MPTCP Path Scheduler to Manage Heterogeneous Paths”. In: *Proceedings of the 13th International Conference on Emerging Networking Experiments and Technologies (CoNEXT ’17)*. ACM, 2017, pp. 147–159. DOI: 10.1145/3143361.3143376.

[25] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. “Neural Adaptive Video Streaming with Pensieve”. In: *Proceedings of the ACM SIGCOMM 2017 Conference*. ACM, 2017, pp. 197–210. DOI: 10.1145/3098822.3098843.

[26] Nick McKeown et al. “OpenFlow: Enabling Innovation in Campus Networks”. In: *ACM SIGCOMM Computer Communication Review* 38.2 (Apr. 2008), pp. 69–74. DOI: 10.1145/1355734.1355746.

[27] Meta Platforms, Inc. *mvfst: An Implementation of the QUIC Transport Protocol*. https://github.com/facebook/mvfst. Accessed: 2025-01-15. 2024.

[28] François Michel et al. “A First Look at Starlink Performance”. In: *Proceedings of the 22nd ACM Internet Measurement Conference (IMC ’22)*. ACM, 2022, pp. 130–136.

[29] Nitinder Mohan et al. “A Multifaceted Look at Starlink Performance”. In: *Proceedings of the ACM Web Conference 2024 (WWW ’24)*. ACM, 2024, pp. 2723–2734. DOI: 10.1145/3589334.3645328.

[30] Suhas Nandakumar et al. *Media over QUIC Transport*. Internet-Draft draft-ietf-moq-transport-16. Work in Progress. Internet Engineering Task Force, Mar. 2025.

[31] Arvind Narayanan et al. “A First Look at Commercial 5G Performance on Smartphones”. In: *Proceedings of The Web Conference 2020 (WWW ’20)*. ACM, 2020, pp. 894–905. DOI: 10.1145/3366423.3380169.

[32] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. “The Akamai Network: A Platform for High-Performance Internet Applications”. In: *ACM SIGOPS Operating Systems Review* 44.3 (2010), pp. 2–19. DOI: 10.1145/1842733.1842736.

[33] Kazuho Oku and Lucas Pardue. *Extensible Prioritization Scheme for HTTP*. RFC 9218. IETF, June 2022. DOI: 10.17487/RFC9218.

[34] Christoph Paasch et al. “Experimental Evaluation of Multipath TCP Schedulers”. In: *Proceedings of the 2014 ACM SIGCOMM Workshop on Capacity Sharing (CSWS ’14)*. ACM, 2014, pp. 27–32.

[35] Jerome H. Saltzer, David P. Reed, and David D. Clark. “End-to-End Arguments in System Design”. In: *ACM Transactions on Computer Systems* 2.4 (Nov. 1984), pp. 277–288. DOI: 10.1145/357401.357402.

[36] Henning Schulzrinne et al. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550. Internet Engineering Task Force, July 2003. DOI: 10.17487/RFC3550.

[37] Heiko Schwarz, Detlev Marpe, and Thomas Wiegand. “Overview of the Scalable Video Coding Extension of the H.264/AVC Standard”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 17.9 (2007), pp. 1103–1120.

[38] Thomas Stockhammer. “Dynamic Adaptive Streaming over HTTP: Standards and Design Principles”. In: *Proceedings of the 2nd Annual ACM Multimedia Systems Conference (MMSys ’11)*. ACM, 2011, pp. 133–144. DOI: 10.1145/1943552.1943572.

[39] Gary J. Sullivan et al. “Overview of the High Efficiency Video Coding (HEVC) Standard”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 22.12 (Dec. 2012), pp. 1649–1668. DOI: 10.1109/TCSVT.2012.2221191.

[40] Haseeb Tanveer et al. “Making Sense of Constellations: Methodologies for Understanding Starlink’s Scheduling Algorithms”. In: *Proceedings of the 2023 ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT ’23) Companion*. ACM, 2023.

[41] Martin Thomson and Sean Turner. *Using TLS to Secure QUIC*. RFC 9001. Internet Engineering Task Force, May 2021. DOI: 10.17487/RFC9001.

[42] Victor Vasiliev. *WebTransport over HTTP/3*. Internet-Draft draft-ietf-webtrans-http3. Work in Progress. Internet Engineering Task Force, 2024.

[43] Thomas Wiegand et al. "Overview of the H.264/AVC Video Coding Standard". In: *IEEE Transactions on Circuits and Systems for Video Technology* 13.7 (July 2003), pp. 560–576. DOI: 10.1109/TCSVT.2003.815165.

A. Reproducibility Guide to the Implementation Repository

The complete source code for the MoMQ prototype is publicly available at `https://github.com/spear-lab/multipath-conferencing-over-quic/tree/optimization`. This appendix highlights only the repository locations needed to understand the MoMQ prototype and rerun the evaluation. The `moq-examples` repository has three layers. It combines thin runtime binaries in `src/bin/`, shared protocol and media logic in `src/`, and reproducibility support in `config/` and `tools/`. Auxiliary documentation, build artifacts, and one-off helpers are omitted on purpose.

Start here.

- `src/bin/tquic_moq_publisher.rs`, `src/bin/tquic_moq_relay.rs`, and `src/bin/tquic_moq_subscriber.rs` are the three runtime entry points. The relay and subscriber are the most important files for the MoMQ rule path and evaluation logic.
- `src/moq_protocol.rs` and `src/mpmoq_protocol.rs` define the base MoQ messages and the prototype control messages added by MoMQ.
- `config/*.yaml` stores the role-specific runtime parameters used to reproduce experiments.
- `tools/run_test*.sh` and `tools/*analysis.py` automate repeated runs and reduce raw outputs into summary figures.

A.1. Key Code Paths

Table A.1 concentrates the repository into the code paths that matter most when reading the implementation together with the thesis.

Table A.1: Repository paths that matter most for understanding and reproducing the prototype.

Concern	Primary paths	Why these files matter
Runtime roles	`src/bin/tquic_moq_publisher.rs`, `src/bin/tquic_moq_relay.rs`, `src/bin/tquic_moq_subscriber.rs`	These are the clearest entry points for the end-to-end system. They cover media publication, relay forwarding, rule handling, subscriber reception, and trace generation.
Protocol and control plane	`src/moq_protocol.rs`, `src/mpmoq_protocol.rs`, `src/moq_crypto.rs`	These files define the wire format and the control messages that carry the MoMQ-specific path and rule semantics described in the design chapters.
Media pipeline	`src/gop_producer.rs`, `src/gop_consumer.rs`, `src/ffmpeg_encoder.rs`, `src/h264_utils.rs`	These files show how raw media is converted into GOP-structured Objects before it enters the QUIC/MoQ transport path.
Receiver and measurements	`src/subscriber_worker.rs`, `src/decoders.rs`, `src/frame_delivery_monitor.rs`, `src/monitoring.rs`	This is the path that reconstructs media, records delivery timing, and writes the CSV outputs used in Chapter 4.
Experiment automation	`config/*.yaml`, `tools/setup_multipath_network.py`, `tools/run_test*.sh`, `tools/*analysis.py`	This layer fixes runtime parameters, sets up controlled multipath topologies, runs repeated experiments, and reduces outputs into tables and figures.

Most readers can ignore `target/`, `doc/`, and the auxiliary `src/bin/analyze_*.rs` or `src/bin/test_*.`

`rs` helpers unless they are extending the prototype rather than reproducing the thesis workflow.

A.2. Minimal Reproduction Workflow

Table A.2 reduces the reproduction process to the concrete files and scripts a reader actually touches. It is intentionally scoped to the thesis workflow rather than to every development utility in the repository.

Table A.2: Minimal workflow for reproducing the prototype evaluation.

Step	Files or scripts	Concrete task and expected output
1	`config/*.yaml`	Edit the role-specific YAML files for the publisher, relay, and subscriber. This is where the experiment selects addresses, interface bindings, QUIC transport settings, scheduler choice, and output locations. At the end of this step, one configuration set fully defines a run.
2	`tools/setup_multipath_network.py`, `tools/multipath_network_config.yaml`, `tools/teardown_multipath_network.py`	If the experiment needs controlled path diversity, create the emulated multipath environment before starting the application roles. The YAML file fixes the topology and impairment parameters. The setup script instantiates the interfaces, and the teardown script removes them after the run. This step is skipped for measurements on the live testbed.
3	`src/bin/tquic_moq_relay.rs`, `src/bin/tquic_moq_subscriber.rs`, `src/bin/tquic_moq_publisher.rs`	Launch the runtime roles in dependency order. Start the relay first, the subscriber second, and the publisher last. The relay receives rule and forwarding state, the subscriber installs the MoMQ logic and records delivery traces, and the publisher injects GOP-structured media Objects into the session. The direct outputs are reconstructed media plus per-run monitoring CSV files.
4	`tools/run_test.sh`, `tools/run_test_heter.sh`, `tools/run_test_homo.sh`	For repeated experiments, use the shell wrappers instead of running single trials manually. These scripts sweep scheduler variants and network templates, repeat runs under the same conditions, and store run-associated outputs together with the configuration used for each trial. This is the path closest to the batch methodology used in Chapter 4.
5	`tools/summarize.py`, `tools/comprehensive_*_analysis.py`, `tools/plot_from_summary.py`	Aggregate the raw CSV outputs into summary statistics and figures. In practice, this stage converts per-run traces into the latency, buffer, and tail-distribution summaries that feed the tables and plots in the evaluation chapter.

This table matches the thesis architecture directly. It separates reusable protocol and media logic in `src/`, thin runtime wrappers in `src/bin/`, and a separate configuration-and-analysis layer for reproducibility. For a reader who only wants the shortest useful path, Steps 3–5 are the core execution chain, while Steps 1–2 determine the exact scenario being reproduced.