Media Over QUIC
Internet-Draft
Intended status: Informational
Expires: 3 September 2026

M. English
L. Pardue
Cloudflare
A. Sharma
Meta
I. Swett
Google

2 March 2026

Denial-of-Service Considerations for Media over QUIC Relay Deployments
draft-englishm-moq-relay-dos-00

Abstract

The Media over QUIC Transport (MoQT) protocol presents denial-of-service risks that differ in character from those facing typical request-response protocols. MoQT relays forward, fan out, and optionally cache media content on behalf of publishers and subscribers. This document complements the MoQT Security Considerations, focusing on the unique considerations for relays.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://englishm.github.io/moq-relay-dos/draft-englishm-moq-relay-dos.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-englishm-moq-relay-dos/.

Discussion of this document takes place on the Media Over QUIC Working Group mailing list (mailto:moq@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/moq/. Subscribe at https://www.ietf.org/mailman/listinfo/moq/.

Source for this draft and an issue tracker can be found at https://github.com/englishm/moq-relay-dos.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Media over QUIC Transport protocol [MOQT] enables media delivery through intermediary relays that forward, fan out, and optionally cache content for subscribers. Unlike request-response protocols where the server's work is roughly proportional to the client's request, MoQT has amplification properties, in the sense that a single SUBSCRIBE can trigger backend lookups, upstream Transport Session establishment, and ongoing media delivery that can significantly exceed the cost of the request itself. This asymmetry is relevant to the denial-of-service risks discussed in this document.

[BCP72] requires that protocol specifications identify potential denial-of-service (DoS) attacks and describe foreseeable risks with due diligence. The MoQT transport specification already addresses resource exhaustion, stream limits, flow control, and some relay-specific concerns. This document provides more detailed analysis of the threat landscape — particularly for relay deployments where the amplification properties of the protocol are most relevant — along with operational guidance for implementers and operators.

1.1. Motivation

This section is to be removed before publishing as an RFC.

This work was motivated by discussion at the MoQ Working Group interim meeting in February 2026. The working group decided that detailed DoS analysis and operational guidance warranted a separate document rather than additions to the transport specification.

1.2. Scope

This document covers denial-of-service and resource exhaustion threats to MoQT relay deployments. Authentication, authorization, and billing mechanisms are out of scope, though they can be critical tools for DoS mitigation in practice and are referenced where relevant. Content filtering mechanisms and their resource costs are also deferred to future work.

Where this document identifies potential changes to the transport protocol that could improve DoS resilience, those are noted but left to proposals against the transport specification. While both this document and the transport specification are under active development, this document also catalogs mechanisms that are currently load-bearing for DoS resilience (Section 8) so that changes elsewhere do not inadvertently weaken existing protections.

1.3. Relationship to the Transport Specification

The MoQT transport specification [MOQT] already covers stream limits, flow control, priority and starvation, timeouts, and some relay-specific concerns (state maintenance, SUBSCRIBE_NAMESPACE with short prefixes). This document does not replace any of that. It adds threat analysis, documents which existing mechanisms are load-bearing for DoS prevention, and provides operational guidance for relay deployments.

2. Conventions and Definitions

This document provides operational guidance for MoQT relay implementations and deployments. It does not modify or override protocol requirements specified in [MOQT].

This document uses the following terms:

Relay: A MoQT intermediary that receives content from publishers and forwards it to subscribers, potentially caching content and serving multiple subscribers from a single upstream subscription.

Control Plane: MoQT operations that establish, modify, and tear down state: SUBSCRIBE, FETCH, PUBLISH_NAMESPACE, SUBSCRIBE_NAMESPACE, REQUEST_UPDATE, TRACK_STATUS, and related messages. These often require coordination with backend systems for routing, authorization, and state management.

Data Plane: MoQT operations involved in delivering media objects to subscribers: QUIC stream management, object framing, and flow control.

3. Threat Model

MoQT relay deployments face resource exhaustion risks from several categories of actors. How severe these risks are in practice depends on the deployment model, authentication requirements, and trust relationships between participants.

3.1. Actors

This section uses "misbehaving" rather than "malicious" to describe problematic actors. A relay's defenses need to handle both deliberate attacks and innocent misconfiguration or buggy implementations — from the relay's perspective, the distinction often does not matter.

Note that a relay acting maliciously with respect to content integrity or confidentiality (interception, manipulation, selective suppression) is a media security concern rather than a DoS concern and is out of scope for this document.

3.1.1. Misbehaving Subscriber

Subscribers tend to face fewer authentication constraints, which can create a broader attack surface. In broadcast scenarios — which are a significant near-term deployment model for MoQT — subscribers may be minimally authenticated or entirely unauthenticated. A malicious subscriber can issue requests that force relays or publishers to commit resources (subscriptions, buffer allocations, backend lookups), manipulate flow control to cause excessive buffering, rapidly create and cancel subscriptions to generate processing overhead, or subscribe to content with no intent to actually consume it.

The impact a misbehaving subscriber can have depends on where they sit in the topology. If subscribers connect through a relay operated by the content provider (or its CDN), the operator can impose limits and terminate abusive sessions with full context about what legitimate behavior looks like. If subscribers connect to third-party relays, the relay may have less information to distinguish attacks from unusual but legitimate usage.

3.1.2. Misbehaving Publisher

Publishers are usually more constrained than subscribers. They tend to have business relationships with the relay operator, face stronger authentication requirements, and their traffic patterns are more predictable and visible. That said, a malicious or compromised publisher can flood namespace advertisement state (forcing relays to propagate and store large numbers of namespace registrations), publish at rates that overwhelm downstream relays and subscribers, or exploit protocol features to force resource commitment without delivering useful content — for example, opening many subgroup streams and keeping them active.

3.1.3. Misbehaving Relay

In relay chains or meshes, a misbehaving relay can cause upstream load through fan-out amplification, create namespace advertisement loops, or fail to enforce resource limits in either direction — failing to protect downstream clients from a misbehaving publisher, or failing to protect upstream publishers from the aggregate load of a large downstream audience. A compromised relay in a chain effectively acts as both a misbehaving subscriber (to its upstream peers) and a misbehaving publisher (to its downstream peers) at the same time.

3.2. Amplification Properties

MoQT has significant amplification properties. A single SUBSCRIBE message can cause a relay to perform backend lookups, establish an upstream subscription to the publisher, and begin forwarding a potentially unbounded stream of media objects. The server-side work can be disproportionately large relative to the effort required to trigger it — potentially much larger than in request-response protocols like HTTP, where the server’s cost is roughly proportional to the request.

This has a direct implication for how limits need to be designed. Per-session limits are necessary but may not be sufficient: if an attacker can open many sessions cheaply, per-session limits just move the problem. Effective protection requires limits scoped to something that is expensive for an attacker to obtain in quantity — an authenticated identity, a billing account, or similar. The specifics of authentication and billing are out of scope here, but session-level controls alone may not be sufficient — they could be one layer of a defense-in-depth strategy.

4. Control Plane Considerations

MoQT relay deployments require authoritative answers to routing questions: which namespaces have been registered via PUBLISH_NAMESPACE, where a subscription should be forwarded, and whether an incoming request is permitted. Every relay — regardless of deployment size — maintains state to answer these questions. The cost of maintaining that state and the consequences of getting it wrong both grow with the scale of the deployment.

On a single relay node, authoritative state is local: lookups and updates are bounded by CPU and memory. As deployments grow to multiple nodes, that state must be shared — through replication, coordination, or some partitioning scheme — and the per-operation cost tends to increase accordingly. MoQT is designed to support distribution at scale through third-party relay networks (see [MOQT], Section 1.1.4), so this document focuses on the cost profile where these pressures are most acute.

In general, operations that read authoritative state tend to be cheaper than operations that write or update it, and this asymmetry tends to widen as the number of nodes sharing that state grows — particularly when those nodes are geographically distributed. Understanding the cost profile of different operations is important for setting appropriate limits. Examining costs through the lens of each control message type is a practical way to do this, particularly for write-heavy operations.

4.1. Authoritative State and Coordination Costs

Several MoQT control messages create or modify authoritative state that the relay uses for routing and resource management:

PUBLISH_NAMESPACE: Registers a namespace as actively published. The relay must record this so that incoming subscriptions can be matched to the correct publisher. In a multi-node deployment, this registration must be visible to other nodes that may receive subscriptions for the same namespace.

SUBSCRIBE: Requests content for a Full Track Name (Track Namespace + Track Name). The relay must look up the registered namespace, determine where the content can be sourced, and may need to establish an upstream forwarding path. Each subscription creates ongoing state that must be maintained for the lifetime of the subscription.

SUBSCRIBE_NAMESPACE: Registers interest in content matching a

namespace prefix. This inverts the namespace-advertisement flow: instead of matching a subscription against registered namespaces, the relay must match incoming PUBLISH_NAMESPACE messages against registered namespace subscriptions. The state cost can be similar to PUBLISH_NAMESPACE but evaluated in the reverse direction.

At the smallest scale, these operations are local lookups and writes. At larger scales, they may require coordination across nodes — whether through consensus protocols, centralized registries, or other mechanisms. The specific costs depend on the deployment architecture, but the relative ordering is consistent: control plane operations that touch authoritative state can be significantly more expensive per message than data plane forwarding. This cost differential means the control plane may warrant particular attention when designing resource exhaustion defenses.

4.2. Rapid Request Cycles

Rapidly creating and canceling subscriptions forces repeated setup and teardown work at the relay — and potentially at upstream publishers — while the attacker pays only the cost of sending small control messages. This is directly analogous to the HTTP/2 "Rapid Reset" attack [HTTP2-RAPID-RESET], where attackers exploited the asymmetry between the cost of sending RST_STREAM and the server-side cost of stream setup.

The risk is not limited to subscription churn. Any control message that triggers work at the relay can be abused through rapid repetition, including priority updates that may force reordering of internal scheduling structures.

Implementations that track request rates per session can detect and terminate sessions exhibiting excessive churn without making progress on useful data transfer.

4.3. Namespace Advertisement Flooding

Because namespace advertisement state may need to be globally registered, an attacker who can publish (or a compromised publisher) can flood the namespace store with PUBLISH_NAMESPACE messages. These are write-heavy operations against authoritative state (see the read/write cost asymmetry discussed in Section 4), and in systems using consensus protocols with per-transaction overhead, a flood of PUBLISH_NAMESPACE messages can quickly exhaust capacity. SUBSCRIBE_NAMESPACE can have a similar cost profile: each namespace subscription registers matching state that must be evaluated against every incoming PUBLISH_NAMESPACE.

Imposing limits on active namespace advertisements and namespace subscriptions — both per session and per authenticated identity — limits the impact a single actor can have on the namespace store.

4.4. Update Coalescing

REQUEST_UPDATE messages have a useful property: unlike HTTP/2 priority tree updates (where each update forces a tree restructure), multiple pending REQUEST_UPDATE messages for the same request can be merged into a single state change before being applied. Rapid REQUEST_UPDATE messages still consume network and parsing resources, but they need not cause proportional backend work.

Coalescing pending updates where semantically equivalent significantly reduces the effectiveness of update-flood attacks.

5. Data Plane Considerations

Data plane resource exhaustion targets the media forwarding path: memory consumed by buffered objects, bandwidth spent on delivery, and processing capacity for framing and scheduling.

5.1. Slow Subscribers

A subscriber that consumes data slower than it is produced causes data to accumulate at the relay. The relay must buffer objects waiting to be read, and that memory pressure could affect other subscribers on the same relay. This can happen because the subscriber's network is slow, because the subscriber's application can't keep up, or because the subscriber is deliberately not consuming data at all.

Relays that detect and handle slow subscribers can limit this accumulation. Options include dropping lower-priority data, canceling subscriptions that fall too far behind the live edge, and imposing per-subscriber buffer limits.

5.2. Join-Time Buffering

When a subscriber joins a live stream and requests historical data — to build a decode buffer or start from a valid random access point — the relay must deliver that historical data while live data continues to arrive. If the subscriber's bandwidth is limited, the historical delivery takes time during which live objects queue at the relay. For high-bitrate streams, this transient buffering spike can be significant.

This is not inherently an attack — it happens in normal operation. But it creates a window that an attacker could exploit by joining many streams simultaneously with constrained receive windows. Limiting the amount of data buffered per subscriber during join operations contains this risk. When limits are exceeded, a relay can cancel the subscription, skip ahead to the live edge, or reduce delivery quality.

5.3. Flow Control Limitations

QUIC flow control (Section 4 of [QUIC]) protects receivers from memory exhaustion, but its protections have limits that are worth understanding:

Within a single MoQT session: QUIC flow control operates at the connection and stream level. MoQT's prioritization mechanism intentionally allows higher-priority subscriptions to consume resources at the expense of lower-priority ones — that is by design for media delivery. But it means a single high-priority subscription can starve other subscriptions within the same session, and QUIC flow control will not prevent it.

Across multiple MoQT sessions: Flow control on one session does not prevent a slow subscriber on that session from causing resource accumulation that affects other sessions at the same relay. A relay must manage cross-session resource allocation independently.

MoQT does not currently provide per-subscription resource isolation. Within a session, a single subscription can consume an unbounded share of buffer memory, outbound bandwidth, and scheduling priority. Across sessions, a single session can similarly dominate relay-wide memory and bandwidth unless the relay imposes application-layer limits. Relays that implement fairness policies across sessions — such as per-session buffer caps or weighted bandwidth allocation — can prevent one subscriber's behavior from degrading service for others.

6. Relay Dual-Role Considerations

Relays are simultaneously subscribers (to upstream publishers) and publishers (to downstream subscribers). This dual role creates DoS considerations that don't apply to original publishers or end subscribers.

6.1. Resource Isolation

A relay serving multiple clients must prevent one client's behavior from degrading service for the others. In practice this means per-client limits on active subscriptions, namespace advertisements, and concurrent streams; independent buffer management per downstream subscription (so a slow subscriber doesn't cause backpressure on other subscribers' data); and fairness policies for shared resources like memory, bandwidth, and backend query capacity.

These two goals sit in tension: the more independently a relay handles each downstream subscription, the less sharing and coalescing it can do upstream, limiting its ability to scale efficiently. Operators need to navigate this tradeoff for their particular deployment.

That said, propagating per-subscriber resource pressure upstream degrades service for every other subscriber on that stream. If one subscriber is slow, the relay needs to handle that locally — dropping data, canceling the subscription, or adjusting delivery — rather than reducing the rate at which it receives data from the publisher.

6.2. Upstream Load Protection

Data plane traffic can aggregate efficiently at relays: multiple downstream subscriptions to the same content collapse into a single upstream subscription, and cached objects are served from local state. Control plane traffic is a different story. If many downstream subscribers simultaneously request historical data, issue REQUEST_UPDATE messages, or churn through subscriptions, the relay may need to forward a high volume of control messages upstream. This fan-in of control traffic from many downstream sessions can overwhelm upstream publishers or relays even when data plane traffic is well-managed.

Effective defenses include rate-limiting control messages forwarded upstream, aggregating or deduplicating upstream requests where possible, and shedding downstream load rather than passing it through to upstream peers.

6.3. Multi-Publisher Fan-Out

When a relay serves content from multiple publishers, subscriber operations can fan out upstream in ways that may not be obvious from the subscriber's perspective. Multi-publisher scenarios introduce additional fan-out and resource isolation considerations that need further analysis. This section will be expanded in future revisions.

7. Operational Recommendations

Appropriate limits depend on deployment architecture, hardware capacity, and expected traffic patterns. The recommendations in this section are expressed as general principles; operators will need to determine specific thresholds for their deployments.

7.1. Cost Tracking

Tracking resource consumption per transport session — and, where possible, per authenticated identity — gives operators the visibility needed to detect abuse. Useful signals include the rate of control messages over time, the number of active subscriptions and namespace advertisements, buffered data per subscriber, and the ratio of control plane activity to useful data transfer.

Sessions or identities that consume disproportionate resources relative to the data transfer they facilitate are candidates for throttling or termination.

7.2. Rate Limiting

Rate limits are the primary defense against control plane flooding. A few properties matter for MoQT relay deployments specifically:

Rate limits are most effective when scoped per-client or per-authenticated-identity, not just globally. A purely global rate limit can be consumed by a single attacker, denying service to everyone else. Scoping limits per deployment unit — per customer account or per content namespace — further isolates abuse so that problems in one context do not affect another.

Per-client limits and systemic circuit breakers serve different purposes and both contribute to a complete defense. Per-client limits prevent individual abuse. Circuit breakers protect the backend when many clients simultaneously generate legitimate but excessive load, as happens during flash crowd events.

When declining individual requests due to overload, explicit protocol feedback (for example, REQUEST_ERROR with EXCESSIVE_LOAD in [MOQT]) is preferable to silent failure. Clear overload signaling helps legitimate peers back off quickly.

Traffic characteristics vary across deployments and evolve over time as usage patterns and hardware capabilities change. Limits calibrated relative to observed legitimate traffic patterns for the deployment tend to be more robust than fixed values.

8. Tracking Protocol Load-Bearing Mitigations

This section is to be removed before publishing as an RFC.

Several MoQT protocol mechanisms serve as mitigations against resource exhaustion, even if some of them were not originally designed with that as their primary purpose. It is worth documenting these explicitly so that proposed changes to the protocol can be evaluated for their security impact, since removing or weakening a mechanism that happens to be load-bearing for DoS prevention could reintroduce risks that were previously mitigated. When evaluating protocol changes, it is worth considering whether existing mitigations are being preserved, replaced with equivalent mechanisms, or inadvertently weakened.

Mechanism	Threat Mitigated	Notes
MAX_REQUEST_ID	Rapid request flooding; unbounded state commitment	Limits outstanding requests a peer can issue. Any proposal to remove or change this needs to provide equivalent protection.
QUIC Stream Limits	Stream exhaustion	Baseline protection. When used as the primary request-limiting mechanism, configuration needs to account for security implications.
QUIC Flow Control	Receiver memory exhaustion	Protects individual receivers but does not provide cross-session fairness or state exhaustion protection.

Subscription Timeouts	Idle resource consumption	Lets relays reclaim resources from inactive subscriptions.
MAX_AUTH_TOKEN_CACHE_SIZE	Memory consumption	Bounds total per-session bytes of registered authorization token aliases/ values
Control message length limit (2^16-1)	Oversized control-message parsing DoS	Bounds the size of each control message

Table 1

9. Security Considerations

This entire document addresses security considerations for MoQT relay deployments. It complements the MoQT Security Considerations. The focus is on denial-of-service threats and resource exhaustion; authentication, authorization, and billing are out of scope but can be essential parts of a complete defense. In particular, the amplification properties described in Section 3.2 mean that effective DoS protection may require per-identity or per-account limits beyond what the transport protocol alone can provide.

10. IANA Considerations

This document has no IANA actions.

11. References

11.1. Normative References

[MOQT] Nandakumar, S., Vasiliev, V., Swett, I., and A. Frindell, "Media over QUIC Transport", Work in Progress, Internet-Draft, draft-ietf-moq-transport-16, 13 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-moq-transport-16>.

[QUIC] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", RFC 9000, DOI 10.17487/RFC9000, May 2021, <https://www.rfc-editor.org/rfc/rfc9000>.

11.2. Informative References

[BCP72] Best Current Practice 72, <https://www.rfc-editor.org/info/bcp72>. At the time of writing, this BCP comprises the following:

Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <https://www.rfc-editor.org/info/rfc3552>.

Gont, F. and I. Arce, "Security Considerations for Transient Numeric Identifiers Employed in Network Protocols", BCP 72, RFC 9416, DOI 10.17487/RFC9416, July 2023, <https://www.rfc-editor.org/info/rfc9416>.

[HTTP2-RAPID-RESET] "HTTP/2 Rapid Reset Attack", 2023, <https://www.cve.org/CVERecord?id=CVE-2023-44487>.

[RFC9775] Perkins, C. S., "IRTF Code of Conduct", RFC 9775, DOI 10.17487/RFC9775, March 2025, <https://www.rfc-editor.org/rfc/rfc9775>.

Contributors

* Ian Swett
* Aman Sharma
* Cullen Jennings
* Will Law
* Luke Curley
* Mike Bishop
* Mo Zanaty

Use of Generative AI

Anthropic’s Claude (Sonnet 4.6 and Opus 4.6) and OpenAI’s GPT-5.3 Codex were used to assist with organizing source materials, structuring the document, and drafting text. All AI-generated content was reviewed and approved by the author. This disclosure follows the guidance in [RFC9775], applied voluntarily to this IETF document.

Authors' Addresses

Mike English
Cloudflare
Email: ietf@englishm.net

Lucas Pardue
Cloudflare
Email: lucas@lucaspardue.com

Aman Sharma
Meta
Email: amsharma@meta.com

Ian Swett
Google
Email: ianswett@google.com