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

A. Frindell
Meta

2 March 2026

QPACK Compression for MoQ Transport
draft-frindell-moq-moqpack-00

Abstract

This document defines an extension to Media over QUIC Transport (MOQT) that enables QPACK compression for control messages. By leveraging QPACK's dynamic table, this extension significantly reduces the overhead of repeated values such as track names and authorization tokens, improving efficiency for sessions with many subscriptions or frequent redundant values.

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://afrind.github.io/draft-frindell-moq-moqpack/draft-frindell-moq-moqpack.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-frindell-moq-moqpack/.

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/afrind/draft-frindell-moq-moqpack.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 4
  1.1. Motivation 4
2. Conventions and Definitions 4
3. Extension Negotiation 5
  3.1. MOQT_QPACK_MAX_TABLE_CAPACITY 5
  3.2. MOQT_QPACK_BLOCKED_STREAMS 5
  3.3. MOQT_QPACK_INDEX_SETUP_AUTH 5
4. QPACK Streams 6
  4.1. Stream Types 6
  4.2. Stream Initialization 7
  4.3. Stream Lifetime 7
5. Compressed Message Formats 7
  5.1. MOQPACK Flag Bit 7
  5.2. Pseudo-Parameter Types 8
    5.2.1. TRACK_NAMESPACE_SET Value Format 8
  5.3. Namespace Reconstruction 9
  5.4. Field Ordering 10
  5.5. Required Fields 10
  5.6. MOQPACK Message Formats 10
    5.6.1. SUBSCRIBE 10
    5.6.2. PUBLISH 11
    5.6.3. FETCH 11
    5.6.4. SUBSCRIBE_NAMESPACE 12
    5.6.5. PUBLISH_NAMESPACE 12
    5.6.6. NAMESPACE 13

5.6.7. NAMESPACE_DONE 13
5.6.8. TRACK_STATUS 13
5.6.9. SUBSCRIBE_OK 13
5.6.10. FETCH_OK 13
5.6.11. Parameter-Only Messages 14
6. QPACK Encoding 15
6.1. Compressed Block Format 15
6.2. Indexing 15
6.3. MOQT Static Table Semantics 16
6.3.1. Field Line Interpretation 16
6.4. Dynamic Table 17
6.4.1. Encoder Stream Instructions 17
6.4.2. Parameter Value Encoding 17
6.4.3. Authorization Token Encoding 18
6.5. Decoding 18
7. Dynamic Table Management 19
7.1. Encoder Behavior 19
7.1.1. Known Received Count 19
7.1.2. Never-Indexed Literals 19
7.1.3. Avoiding Flow Control Deadlocks 20
7.2. Decoder Behavior 20
7.2.1. Decoder Instructions with Request IDs 20
8. Error Handling 21
8.1. MOQPACK_DECOMPRESSION_FAILED 21
8.2. QPACK Errors 21
9. Security Considerations 21
9.1. Dynamic Table State 21
9.2. Compression Oracle Attacks 22
9.3. Resource Exhaustion 22
10. IANA Considerations 22
10.1. Setup Option Types 22
10.2. Unidirectional Stream Types 22
10.3. Pseudo-Parameter Types 23
10.4. Message Types 23
10.5. Session Error Codes 24
11. References 24
11.1. Normative References 25
11.2. Informative References 25
Acknowledgments 25
Example Encoding 25
Scenario 26
First SUBSCRIBE 26
Subsequent SUBSCRIBEs to Same Track 27
SUBSCRIBE to Different Track, Same Namespace 27
Code Point Summary 28
Setup Options 28
Stream Types 28
Pseudo-Parameter Types 29

MOQPACK Message Types 29
QPACK Library Adaptation Notes 30
Static Table 30
Prohibited Encodings 31
Encoder Stream Instructions 31
Decoder Stream: Request IDs Instead of Stream IDs 31
Entry Size Calculation 32
Author's Address 32

1. Introduction

Media over QUIC Transport (MOQT) [MOQT] control message fields and parameters can contain large values that are repeated across many messages within a session. The base MOQT specification transmits this information in full each time it appears, which can result in significant overhead.

This document defines an extension that uses QPACK [QPACK] to compress MOQT message parameters. QPACK provides:

* Dynamic table for referencing previously transmitted values

* Static table with pre-defined common values

* Stream blocking semantics suitable for QUIC

By treating MOQT parameters as QPACK field lines, this extension enables efficient compression of repeated values while maintaining compatibility with QPACK's existing infrastructure.

1.1. Motivation

Consider a session where a client sends 100 SUBSCRIBE messages, each carrying the same 500-byte authorization token. Without compression, this results in 50,000 bytes of token data. With QPACK compression, the token is transmitted once and subsequent references require only a few bytes, reducing total overhead to approximately 600 bytes.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The terms "endpoint", "session", "publisher", and "subscriber" are defined in [MOQT].

3. Extension Negotiation

This extension is negotiated during MOQT session establishment using Setup Options. CLIENT_SETUP and SERVER_SETUP messages always use standard MOQT encoding and are never MOQPACK compressed.

3.1. MOQT_QPACK_MAX_TABLE_CAPACITY

The MOQT_QPACK_MAX_TABLE_CAPACITY setup option (Option Type 0x10) specifies the maximum size in bytes of the QPACK dynamic table the endpoint is willing to maintain for decoding. This corresponds to SETTINGS_QPACK_MAX_TABLE_CAPACITY in HTTP/3.

The value is encoded as a variable-length integer. The default value is 0.

QPACK compression is enabled when both endpoints send this parameter with a value greater than 0. If either endpoint omits this parameter or sends a value of 0, QPACK compression MUST NOT be used and control messages use the standard MOQT format.

3.2. MOQT_QPACK_BLOCKED_STREAMS

The MOQT_QPACK_BLOCKED_STREAMS setup option (Option Type 0x11) specifies the maximum number of streams that can be blocked waiting for dynamic table updates. This corresponds to SETTINGS_QPACK_BLOCKED_STREAMS in HTTP/3.

The value is encoded as a variable-length integer. The default value is 0, which prevents any stream from being blocked. When set to 0, encoders MUST NOT reference dynamic table entries that have not been acknowledged.

This option is only meaningful when QPACK compression is enabled.

3.3. MOQT_QPACK_INDEX_SETUP_AUTH

The MOQT_QPACK_INDEX_SETUP_AUTH setup option (Option Type 0x12) controls whether AUTHORIZATION TOKEN options from the Setup messages are implicitly inserted into the dynamic table.

The value is encoded as a variable-length integer:

* 0: Do not implicitly insert setup auth tokens (default)

* 1: Implicitly insert setup auth tokens

When either endpoint omits this option or sends 0, implicit insertion does not occur and endpoints that wish to reference auth tokens explicitly insert them via the encoder stream.

This allows endpoints to authenticate the connection via setup auth tokens while still using Never-Indexed Literals for subsequent auth token references if desired.

When both endpoints send MOQT_QPACK_INDEX_SETUP_AUTH with value 1, any AUTHORIZATION TOKEN options from the Setup messages are implicitly inserted into the dynamic table without requiring encoder stream instructions.

Tokens from the Setup message, in the order they appeared, are inserted into the receiver's decoder dynamic table (indices 0, 1, 2, ...)

This allows the client to immediately reference its setup auth token in the first SUBSCRIBE message using a dynamic table reference, without resending it on the encoder stream.

The implicit entries count against the MOQT_QPACK_MAX_TABLE_CAPACITY limit. If the implicit entries would exceed the peer's advertised capacity, the excess entries (in reverse order) are not inserted and cannot be referenced.

Encoder stream insertions after setup use indices starting after the implicit entries. For example, if CLIENT_SETUP contained 2 auth tokens, the first explicit insertion would be at index 2.

4. QPACK Streams

When QPACK compression is negotiated, each endpoint opens two unidirectional streams for QPACK signaling.

4.1. Stream Types

This extension defines two new MOQT unidirectional stream types:

QPACK_ENCODER_STREAM (0x1f107a60): Carries QPACK encoder instructions from the endpoint that opens the stream. The format and semantics are defined in Section 4.3.1 of [QPACK].

QPACK_DECODER_STREAM (0x1f107a61): Carries QPACK decoder instructions from the endpoint that opens the stream. The format and semantics are defined in Section 4.4.1 of [QPACK].

4.2. Stream Initialization

Each endpoint MUST open exactly one QPACK encoder stream and one QPACK decoder stream for QPACK use. These streams MUST be opened before sending any message with a Compressed Block.

An endpoint MAY open these streams immediately after sending its Setup message if it included MOQT_QPACK_MAX_TABLE_CAPACITY with a non-zero value. If a receiving endpoint does not enable QPACK (omits the parameter or sends value 0), it MAY send STOP_SENDING on these streams; this is not an error and the streams SHOULD be reset.

Note that implicitly-inserted dynamic table entries from Setup auth tokens (see Section 3.3) do not require encoder stream instructions. A Compressed Block MAY reference these implicit entries even if no encoder stream instructions have been sent.

If an endpoint receives a Compressed Block that references a dynamic table entry beyond the implicit entries before receiving any encoder stream data, it MUST buffer the message until the required encoder instructions arrive or close the session with PROTOCOL_VIOLATION if buffering limits are exceeded.

4.3. Stream Lifetime

QPACK encoder and decoder streams MUST remain open for the duration of the MOQT session. If either stream is closed after QPACK is negotiated, the endpoint MUST close the MOQT session with PROTOCOL_VIOLATION.

5. Compressed Message Formats

5.1. MOQPACK Flag Bit

This extension uses a flag bit in the message type to indicate if QPACK is used. Bit 6 (0x40) of the message type indicates MOQPACK format:

* Type & 0x40 == 0: Standard MOQT format

* Type & 0x40 == 0x40: MOQPACK format

For example: - SUBSCRIBE standard = 0x03 - SUBSCRIBE MOQPACK = 0x43

All MOQPACK message types listed in this document are reserved in the MOQT message type registry and MUST NOT be used for other purposes.

When MOQPACK is negotiated, endpoints MUST accept both standard and MOQPACK formats for all applicable messages. An endpoint MAY send either format, but SHOULD prefer MOQPACK format to benefit from compression.

When MOQPACK is NOT negotiated, endpoints MUST NOT send MOQPACK format messages and MUST close the session with PROTOCOL_VIOLATION if they receive one.

In MOQPACK format, Track Namespace, Track Name, and Parameters are moved into a QPACK Compressed Block. Other message-specific fields including Properties remain unchanged.

When the Compressed Block is the last field in the message, it extends to the end of the message payload (as determined by the message Length field) and no explicit Compressed Block Length is needed. When additional fields follow the Compressed Block (such as Properties), an explicit Compressed Block Length field is present to delimit the block.

5.2. Pseudo-Parameter Types

The following pseudo-parameter types are reserved for encoding namespace and track name fields in the Compressed Block:

Type	Name	Description
0x0A	TRACK_NAMESPACE_ELEMENT	Single element of a namespace tuple
0x0B	TRACK_NAMESPACE_SET	Full serialized namespace tuple
0x0C	TRACK_NAME	Track Name

Table 1

These pseudo-types use the same encoding as regular parameters: Literal with Static Name Reference for new values, or Indexed with Dynamic Table for previously-inserted values.

5.2.1. TRACK_NAMESPACE_SET Value Format

The value of a TRACK_NAMESPACE_SET field uses the standard MOQT Track Namespace serialization as defined in [MOQT]:

```
TRACK_NAMESPACE_SET Value {
  Number of Track Namespace Fields (vi64),
  Track Namespace Field (..) ...
}

Track Namespace Field {
  Track Namespace Field Length (vi64),
  Track Namespace Field Value (..)
}
```

Each Track Namespace Field Value MUST contain at least one byte, consistent with the requirement in [MOQT].

5.3. Namespace Reconstruction

A Track Namespace, Track Namespace Prefix or Track Namespace Suffix is reconstructed by assembling consecutive TRACK_NAMESPACE_ELEMENT and TRACK_NAMESPACE_SET field lines, which MUST appear first in the Compressed Block before TRACK_NAME or any parameters.

TRACK_NAMESPACE_ELEMENT adds a single element to the namespace tuple. TRACK_NAMESPACE_SET appends all elements from a serialized tuple. These can be intermixed and appear in any combination. For example:

Namespace ("conference", "room1", "audio"):

Option A - All elements:
  TRACK_NAMESPACE_ELEMENT: "conference"
  TRACK_NAMESPACE_ELEMENT: "room1"
  TRACK_NAMESPACE_ELEMENT: "audio"

Option B - Full set:
  TRACK_NAMESPACE_SET: ("conference", "room1", "audio")

Option C - Mixed:
  TRACK_NAMESPACE_ELEMENT: "conference"
  TRACK_NAMESPACE_SET: ("room1", "audio")

This allows encoders to maximize compression by inserting commonly-reused elements or partial tuples into the dynamic table.

The message type determines the semantics of the assembled namespace:

* SUBSCRIBE, PUBLISH, FETCH, TRACK_STATUS, PUBLISH_NAMESPACE: Full namespace

* SUBSCRIBE_NAMESPACE: Namespace prefix

* NAMESPACE, NAMESPACE_DONE: Namespace suffix

5.4. Field Ordering

Namespace elements (TRACK_NAMESPACE_ELEMENT and TRACK_NAMESPACE_SET) MUST appear first in the Compressed Block, followed by TRACK_NAME (if present), followed by parameters in increasing order of parameter type.

Within the namespace elements section, entries appear in the order they contribute to the namespace tuple and are not required to be in increasing order of type. TRACK_NAMESPACE_ELEMENT (0x0A) and TRACK_NAMESPACE_SET (0x0B) MAY be intermixed.

Parameters (excluding pseudo-parameter types) MUST appear in increasing order of their parameter type. If parameters appear out of order, the receiver MUST close the session with PROTOCOL_VIOLATION.

5.5. Required Fields

When decoding a Compressed Block, the receiver MUST verify that all required fields are present:

* SUBSCRIBE, PUBLISH, TRACK_STATUS, Standalone FETCH: At least one namespace element (TRACK_NAMESPACE_ELEMENT or TRACK_NAMESPACE_SET) and TRACK_NAME

* SUBSCRIBE_NAMESPACE, PUBLISH_NAMESPACE, NAMESPACE, NAMESPACE_DONE: At least one namespace element

* Joining FETCH, parameter-only messages: No required pseudo-parameters

An empty namespace (zero elements) is valid only if explicitly allowed by the message semantics.

If a required field is missing, the receiver MUST close the session with PROTOCOL_VIOLATION.

5.6. MOQPACK Message Formats

5.6.1. SUBSCRIBE

```
SUBSCRIBE Message (MOQPACK) {
  Type (vi64) = 0x43,
  Length (16),
  Request ID (vi64),
  Track Alias (vi64),
  Compressed Block (..)
}
```

The Compressed Block contains namespace elements (TRACK_NAMESPACE_ELEMENT and/or TRACK_NAMESPACE_SET), TRACK_NAME (0x0C), and any parameters (AUTHORIZATION_TOKEN, SUBSCRIBER_PRIORITY, etc.).

5.6.2. PUBLISH

```
PUBLISH Message (MOQPACK) {
  Type (vi64) = 0x5D,
  Length (16),
  Request ID (vi64),
  Track Alias (vi64),
  Compressed Block Length (vi64),
  Compressed Block (..),
  Properties (..)
}
```

The Compressed Block contains namespace elements, TRACK_NAME, and any parameters. Properties remain outside the compressed block and use standard MOQT encoding, including IMMUTABLE_EXTENSIONS which are not QPACK compressed.

5.6.3. FETCH

```
Standalone Fetch (MOQPACK) {
  Start Location (Location),
  End Location (Location),
  Compressed Block (..)
}

Joining Fetch (MOQPACK) {
  Joining Request ID (vi64),
  Join Type (vi64),
  Joining Start (vi64),
  Compressed Block (..)
}

FETCH Message (MOQPACK) {
  Type (vi64) = 0x56,
  Length (16),
  Request ID (vi64),
  Fetch Type (vi64),
  [Standalone (Standalone Fetch QPACK),]
  [Joining (Joining Fetch QPACK),]
}
```

For Standalone Fetch, the Compressed Block contains namespace elements, TRACK_NAME, and parameters. For Joining Fetch, the Compressed Block contains only parameters (the track is inherited from the joined subscription).

5.6.4. SUBSCRIBE_NAMESPACE

```
SUBSCRIBE_NAMESPACE Message (MOQPACK) {
  Type (vi64) = 0x51,
  Length (16),
  Request ID (vi64),
  Subscribe Options (vi64),
  Compressed Block (..)
}
```

The Compressed Block contains namespace prefix elements and any parameters.

5.6.5. PUBLISH_NAMESPACE

```
PUBLISH_NAMESPACE Message (MOQPACK) {
  Type (vi64) = 0x46,
  Length (16),
  Request ID (vi64),
  Compressed Block (..)
}
```

The Compressed Block contains namespace elements and any parameters.

5.6.6. NAMESPACE

```
NAMESPACE Message (MOQPACK) {
  Type (vi64) = 0x48,
  Length (16),
  Compressed Block (..)
}
```

The Compressed Block contains namespace suffix elements. This message has no parameters.

5.6.7. NAMESPACE_DONE

```
NAMESPACE_DONE Message (MOQPACK) {
  Type (vi64) = 0x4E,
  Length (16),
  Compressed Block (..)
}
```

The Compressed Block contains namespace suffix elements. This message has no parameters.

5.6.8. TRACK_STATUS

TRACK_STATUS uses the same format as SUBSCRIBE but with type 0x4D. The Compressed Block contains namespace elements, TRACK_NAME, and any applicable parameters.

5.6.9. SUBSCRIBE_OK

```
SUBSCRIBE_OK Message (MOQPACK) {
  Type (vi64) = 0x44,
  Length (16),
  Request ID (vi64),
  Compressed Block Length (vi64),
  Compressed Block (..),
  Properties (..)
}
```

The Compressed Block contains only parameters. Properties remain outside the Compressed Block and use standard MOQT encoding. The Compressed Block Length is required because Properties consume the remainder of the message.

5.6.10. FETCH_OK

```
FETCH_OK Message (MOQPACK) {
  Type (vi64) = 0x58,
  Length (16),
  Request ID (vi64),
  Compressed Block Length (vi64),
  Compressed Block (..),
  Properties (..)
}
```

The Compressed Block contains only parameters. Properties remain outside the Compressed Block and use standard MOQT encoding. The Compressed Block Length is required because Properties consume the remainder of the message.

5.6.11. Parameter-Only Messages

The following messages have parameters but no namespace, track name, or trailing properties. When MOQPACK is negotiated, these messages MAY use MOQPACK format with the 0x40 flag bit set. The MOQPACK format replaces the standard Parameters field with a Compressed Block that consumes the remainder of the message:

Standard Type	MOQPACK Type	Message
0x02	0x42	REQUEST_UPDATE
0x05	0x45	REQUEST_ERROR
0x07	0x47	REQUEST_OK
0x1E	0x5E	PUBLISH_OK

Table 2

```
Parameter-Only Message (MOQPACK) {
  Type (vi64) = <standard type> | 0x40,
  Length (16),
  [Message-specific fields...],
  Compressed Block (..)
}
```

The Compressed Block contains only parameters (no pseudo-parameter types). Message-specific fields (Request ID, error codes, etc.) remain unchanged.

6. QPACK Encoding

This extension uses QPACK's wire encoding formats exactly as specified in [QPACK]. The only difference is the interpretation of static table references: instead of indexing into the HTTP static table of string name-value pairs, the static table index IS the MOQT parameter type.

This allows existing QPACK encoder/decoder implementations to be reused with minimal modification.

6.1. Compressed Block Format

Each Compressed Block begins with the standard QPACK encoded field section prefix as defined in Section 4.5.1 of [QPACK]:

```
Compressed Block {
  Required Insert Count (8+),
  Sign and Delta Base (8+),
  Encoded Field Lines (..)
}
```

Required Insert Count: Encoded as specified in Section 4.5.1.1 of [QPACK]. Indicates the minimum dynamic table state needed to decode this block. A value of 0 means the block has no dynamic table references.

Base: Encoded as a sign bit and Delta Base as specified in Section 4.5.1.2 of [QPACK]. Used to resolve relative indices in field line representations.

6.2. Indexing

Dynamic table references in field lines use relative indexing as specified in [QPACK] Section 3.2.5. A relative index of 0 refers to the entry with absolute index equal to Base - 1. Encoders and decoders MUST use relative indices, not absolute indices, in Compressed Blocks.

Post-Base indexing (Section 3.2.6 of [QPACK]) MAY be used for entries inserted after the Base. This enables single-pass encoding where the encoder inserts entries while encoding a field section and references them using Post-Base indices.

6.3. MOQT Static Table Semantics

In standard QPACK, a static table index retrieves a predefined (name, value) pair. In this extension, the static table conceptually contains entries where the "name" is the parameter type integer and there is no predefined value.

The static table index equals the MOQT parameter type:

For example: * Static index 0x02 represents DELIVERY_TIMEOUT * Static index 0x03 represents AUTHORIZATION_TOKEN * Static index 0x0A represents TRACK_NAMESPACE_ELEMENT * Static index 0x0C represents TRACK_NAME * Static index 0x20 represents SUBSCRIBER_PRIORITY

This means any valid MOQT parameter type can be referenced by static index without requiring pre-registration in a table.

6.3.1. Field Line Interpretation

In QPACK field line encodings, the T bit selects between static table (T=1) and dynamic table (T=0). Since MOQT parameter types are integers that map directly to static table indices, the following encodings are used:

Literal Field Line With Static Name Reference (T=1): The Name Index is the MOQT parameter type. The Value field contains the parameter value. Use this to send a parameter value.

Indexed Field Line with Dynamic Table (T=0): References the dynamic table using a relative index. The retrieved entry contains a complete MOQT parameter (type and value).

Indexed Field Line with Post-Base Index: References a dynamic table entry inserted after the Base. Used for single-pass encoding. See [QPACK] Section 4.5.3.

The following QPACK field line encodings are prohibited:

Indexed Field Line with Static Table (T=1): PROHIBITED. The MOQT static table has no predefined values.

Literal Field Line With Dynamic Name Reference (T=0): PROHIBITED. Parameter types are always known integers; there is no need to reference the dynamic table for a parameter type.

Literal Field Line with Post-Base Name Reference: PROHIBITED. Parameter types are always known integers; there is no need to reference the dynamic table for a parameter type.

Literal Field Line With Literal Name: PROHIBITED. Parameter types are integers, never string literals.

Huffman-encoded string literals (H=1): PROHIBITED. The CPU cost outweighs the minimal space savings for typical MOQT values. Encoders MUST set the H bit to 0 for all string literals.

Receivers MUST treat prohibited encodings as a PROTOCOL_VIOLATION.

6.4. Dynamic Table

Dynamic table entries store complete MOQT parameters (type and value). The entry size calculation follows [QPACK] Section 3.2.1: the size of an entry is the sum of its name size, value size, and 32 bytes of overhead. This extension uses a fixed name size of 4 bytes for the parameter type regardless of its encoded length.

6.4.1. Encoder Stream Instructions

Set Dynamic Table Capacity: Sets the dynamic table capacity up to the peer's MOQT_QPACK_MAX_TABLE_CAPACITY. Encoders MAY reduce capacity dynamically. See [QPACK] Section 4.3.1.

Insert With Static Name Reference (T=1): The Name Index is the MOQT parameter type. Inserts a new dynamic table entry with that parameter type and the provided value.

Duplicate: Duplicates an existing dynamic table entry at a new index. Useful when an entry is near eviction but still frequently referenced. See [QPACK] Section 4.3.4.

Insert With Dynamic Name Reference (T=0): PROHIBITED. Parameter types are always known integers.

Insert With Literal Name: PROHIBITED. Parameter types are integers, never strings.

Receivers MUST close the session with PROTOCOL_VIOLATION if a prohibited encoder instruction is received.

6.4.2. Parameter Value Encoding

QPACK values contain the raw parameter value bytes without any MOQT length prefix. The QPACK value length field serves as the length for binary values.

For binary parameters (odd parameter types in MOQT): The QPACK value

contains the raw binary bytes. The QPACK value length replaces the MOQT Length field. No length prefix is included in the value.

For integer parameters (even parameter types in MOQT): The QPACK value contains the varint-encoded integer. Note that this encoding carries redundant length information: the QPACK value length specifies the byte count, while the varint encoding is self-delimiting. If the varint-encoded length does not match the QPACK value length, the receiver MUST close the session with PROTOCOL_VIOLATION.

For example, DELIVERY_TIMEOUT with value 200: ~~~ QPACK Value Length: 2 QPACK Value: 0xC8 0x01 (varint encoding of 200) ~~~

The receiver decodes the varint and verifies it consumed exactly 2 bytes.

6.4.3. Authorization Token Encoding

The AUTHORIZATION TOKEN parameter value contains the Token structure:

Token Value = Token Type (vi64) || Token Payload (..)

For example, a JWT token (Token Type 1) is encoded as:

Literal Field Line With Name Reference:
  Name Index: 0x03 (AUTHORIZATION_TOKEN)
  Value: 0x01 || "eyJhbGciOiJIUzI1NiIs..."

When this token is inserted into the dynamic table, subsequent references use only a single-byte Indexed Field Line.

6.5. Decoding

When receiving a message with a Compressed Block, the receiver:

1. Parses fixed message fields (Request ID, Track Alias, etc.)

2. If a Compressed Block is present: a. Waits for any referenced dynamic table entries to become available, subject to MOQT_QPACK_BLOCKED_STREAMS limits b. Decodes the QPACK Compressed Block to recover namespace elements, track name, and parameters

3. Processes the fully decoded message

If QPACK decoding fails, the receiver MUST close the session with MOQPACK_DECOMPRESSION_FAILED.

The total size of the decompressed message fields (the sum of all parameter values, namespace elements, and track name, excluding interior length fields) MUST NOT exceed 65535 bytes. If a decompressed message exceeds this limit, the receiver MUST close the session with MOQPACK_DECOMPRESSION_FAILED.

7. Dynamic Table Management

7.1. Encoder Behavior

Encoders SHOULD insert frequently-used parameter values into the dynamic table. Authorization tokens, Track Namespace Elements and Track names that will be reused across multiple messages are prime candidates for insertion.

Short values like brief track names ("audio", "video") MAY be sent as literals rather than inserted, since the overhead of insertion and indexed reference is similar to sending the literal value. Insertion is more beneficial for: - Long values (large auth tokens, long namespace elements) - Values that will be reused across multiple messages (namespace elements shared by many tracks, auth tokens used for many subscriptions)

Encoders MUST respect the peer’s MOQT_QPACK_MAX_TABLE_CAPACITY and MOQT_QPACK_BLOCKED_STREAMS limits when making insertion and reference decisions.

Encoders SHOULD use the QPACK duplicate instruction when a dynamic table entry is at risk of eviction but is still frequently referenced.

7.1.1. Known Received Count

Encoders track the Known Received Count as specified in [QPACK] Section 2.1.4 to determine which entries can be referenced without blocking. Encoders MUST only reference dynamic table entries with absolute index less than the Known Received Count when the number of streams that would be blocked by the reference equals MOQT_QPACK_BLOCKED_STREAMS.

7.1.2. Never-Indexed Literals

The ’N’ bit in Literal Field Line representations signals that a value MUST NOT be indexed by intermediaries. Encoders MAY set the ’N’ bit for sensitive values when: - The value should not be cached by relays - The value has low entropy and is vulnerable to compression attacks

Note that when MOQT_QPACK_INDEX_SETUP_AUTH is enabled, setup auth tokens are implicitly inserted into the dynamic table (see Section 3.3). Endpoints that require 'N' bit semantics for auth tokens MUST NOT enable MOQT_QPACK_INDEX_SETUP_AUTH and SHOULD instead send tokens as Never-Indexed Literals in each message.

Intermediaries that re-encode MOQT messages MUST preserve the 'N' bit semantics: values encoded with N=1 MUST NOT be inserted into the dynamic table.

7.1.3. Avoiding Flow Control Deadlocks

Writing large encoder instructions can cause deadlocks if the decoder withholds flow control credit until the instruction is complete. To avoid this, encoders SHOULD NOT write an encoder instruction unless sufficient stream and connection flow-control credit is available for the entire instruction. If sufficient credit is not available, encoders SHOULD use literal encodings instead. See [QPACK] Section 2.1.3.

7.2. Decoder Behavior

Decoders MUST process encoder instructions from the QPACK encoder stream before processing any message that might reference those insertions.

Decoders MUST send Section Acknowledgment instructions on the QPACK decoder stream after successfully decoding a Compressed Block that references the dynamic table (Required Insert Count > 0). Compressed Blocks that contain only Literal Field Lines with Static Name References do not require acknowledgment.

7.2.1. Decoder Instructions with Request IDs

QPACK decoder instructions that reference streams (Section Acknowledgment, Stream Cancellation) use Request IDs instead of QUIC Stream IDs. This allows MOQT to operate over WebTransport where QUIC Stream IDs are not exposed to the application.

Section Acknowledgment: Carries a Request ID. Acknowledges successful decoding of a Compressed Block that referenced the dynamic table. For streams with multiple such Compressed Blocks (e.g., SUBSCRIBE_NAMESPACE response stream with multiple NAMESPACE messages referencing the dynamic table), successive Section Acknowledgments for the same Request ID acknowledge successive Compressed Blocks in the order they were sent. Compressed Blocks with no dynamic table references are not counted.

Stream Cancellation: Carries a Request ID. Indicates the stream was cancelled before the Compressed Block(s) could be decoded. The encoder MUST NOT count references from that stream when determining eviction eligibility.

Insert Count Increment: Unchanged from [QPACK]. Carries an increment value, no Request ID.

The encoder tracks how many Compressed Blocks it has sent for each Request ID. When it receives a Section Acknowledgment for a Request ID, it knows which Compressed Block was acknowledged based on the count.

8. Error Handling

8.1. MOQPACK_DECOMPRESSION_FAILED

This document defines a new MOQT session error code:

MOQPACK_DECOMPRESSION_FAILED (0xTBD): A QPACK Compressed Block could not be decoded, or the decompressed message exceeded implementation limits. This is always a session error; the endpoint MUST close the MOQT session.

8.2. QPACK Errors

QPACK decoding errors (as defined in Section 6 of [QPACK]) result in session termination. The specific mapping is:

QPACK Error	MOQT Session Error
QPACK_DECOMPRESSION_FAILED	MOQPACK_DECOMPRESSION_FAILED
QPACK_ENCODER_STREAM_ERROR	PROTOCOL_VIOLATION
QPACK_DECODER_STREAM_ERROR	PROTOCOL_VIOLATION

Table 3

9. Security Considerations

9.1. Dynamic Table State

The QPACK dynamic table maintains state across messages. An attacker with knowledge of dynamic table contents could potentially:

* Determine which authorization tokens have been used

* Infer subscription patterns from parameter compression ratios

Implementations SHOULD consider these privacy implications when deciding which values to insert into the dynamic table.

9.2. Compression Oracle Attacks

As with HTTP compression, implementers need to take care to avoid compression oracle attacks where an attacker can infer secret values by observing compressed message sizes. Applications SHOULD NOT mix attacker-controlled data with secret authorization tokens in the same field section.

9.3. Resource Exhaustion

Endpoints MUST enforce the negotiated table capacity limits to prevent resource exhaustion attacks. An endpoint that attempts to exceed these limits causes a session error.

10. IANA Considerations

10.1. Setup Option Types

This document registers the following Setup Option Types in the "MOQT Setup Options" registry:

Parameter Type	Parameter Name	Specification
0x10	MOQT_QPACK_MAX_TABLE_CAPACITY	Section 3
0x11	MOQT_QPACK_BLOCKED_STREAMS	Section 3
0x12	MOQT_QPACK_INDEX_SETUP_AUTH	Section 3

Table 4

10.2. Unidirectional Stream Types

This document registers the following unidirectional stream types in the "MOQT Stream Types" registry:

Stream Type	Name	Specification
0x1f107a60	QPACK_ENCODER_STREAM	Section 4.1
0x1f107a61	QPACK_DECODER_STREAM	Section 4.1

Table 5

10.3. Pseudo-Parameter Types

This document registers the following pseudo-parameter types in the "MOQT Message Parameters" registry. These types are reserved for use in QPACK Compressed Blocks and MUST NOT appear in standard Parameters fields.

Parameter Type	Parameter Name	Specification
0x0A	TRACK_NAMESPACE_ELEMENT	Section 5.2
0x0B	TRACK_NAMESPACE_SET	Section 5.2
0x0C	TRACK_NAME	Section 5.2

Table 6

10.4. Message Types

This document reserves the following message types in the "MOQT Message Types" registry for use as MOQPACK-format messages:

Message Type	Name	Specification
0x42	MOQPACK REQUEST_UPDATE	Section 5.6
0x43	MOQPACK SUBSCRIBE	Section 5.6
0x44	MOQPACK SUBSCRIBE_OK	Section 5.6
0x45	MOQPACK REQUEST_ERROR	Section 5.6
0x46	MOQPACK PUBLISH_NAMESPACE	Section 5.6
0x47	MOQPACK REQUEST_OK	Section 5.6
0x48	MOQPACK NAMESPACE	Section 5.6
0x4D	MOQPACK TRACK_STATUS	Section 5.6
0x4E	MOQPACK NAMESPACE_DONE	Section 5.6
0x51	MOQPACK SUBSCRIBE_NAMESPACE	Section 5.6
0x56	MOQPACK FETCH	Section 5.6
0x58	MOQPACK FETCH_OK	Section 5.6
0x5D	MOQPACK PUBLISH	Section 5.6
0x5E	MOQPACK PUBLISH_OK	Section 5.6

Table 7

10.5. Session Error Codes

This document registers the following session error code in the "MOQT Session Error Codes" registry:

Error Code	Name	Specification
0xTBD	MOQPACK_DECOMPRESSION_FAILED	Section 8.1

Table 8

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>.

[QPACK] Krasic, C., Bishop, M., and A. Frindell, Ed., "QPACK: Field Compression for HTTP/3", RFC 9204, DOI 10.17487/RFC9204, June 2022, <https://www.rfc-editor.org/rfc/rfc9204>.

[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>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

11.2. Informative References

[HPACK] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <https://www.rfc-editor.org/rfc/rfc7541>.

Acknowledgments

The QPACK specification [QPACK] provides the foundation for this work. The design of HTTP/3 header compression informed many decisions in this document.

Claude, by Anthropic, assisted in drafting this document.

Example Encoding

This appendix provides an example of QPACK-compressed MOQT messages, demonstrating the Required Insert Count, Base, and relative indexing.

Scenario

A client sends three SUBSCRIBE messages to the same track with the same authorization token:

* Track Namespace: ("conference", "room42")
* Track Name: "audio"
* Token Type: 1 (e.g., JWT)
* Token Value: "eyJhbGciOiJIUzI1NiIs..." (500 bytes)

The auth token was sent in CLIENT_SETUP and is implicitly inserted at dynamic table absolute index 0 after QPACK negotiation succeeds.

First SUBSCRIBE

The encoder inserts namespace elements on the encoder stream. The short track name "audio" is sent as a literal rather than inserted:

```
Encoder Stream:
  Insert With Static Name Reference
    Name Index: 0x0A (TRACK_NAMESPACE_ELEMENT)
    Value: "conference"
  Insert With Static Name Reference
    Name Index: 0x0A (TRACK_NAMESPACE_ELEMENT)
    Value: "room42"
```

Dynamic table state after insertions:

Absolute Index	Parameter Type	Value
0	AUTH_TOKEN (implicit)	Token Type 1, "eyJ..."
1	TRACK_NAMESPACE_ELEMENT	"conference"
2	TRACK_NAMESPACE_ELEMENT	"room42"

Table 9

The encoder sends the SUBSCRIBE with Required Insert Count = 3 and Base = 3:

```
SUBSCRIBE Message:
  Type: 0x43
  Request ID: 1
  Track Alias: 100
  Compressed Block:
    Required Insert Count: 3 (encoded per RFC 9204 Section 4.5.1.1)
    Base: Sign=0, Delta=0 (Base = Required Insert Count = 3)
    Indexed Field Line (Dynamic, relative index 1)  // abs 1 = "conference"
    Indexed Field Line (Dynamic, relative index 0)  // abs 2 = "room42"
    Literal Field Line (Static Name 0x0C, Value "audio")  // TRACK_NAME
    Indexed Field Line (Dynamic, relative index 2)  // abs 0 = AUTH_TOKEN
```

Relative index calculation: relative = Base - 1 - absolute

* "conference": relative = 3 - 1 - 1 = 1

* "room42": relative = 3 - 1 - 2 = 0

* AUTH_TOKEN: relative = 3 - 1 - 0 = 2

Compressed Block: ~12 bytes (including prefix and literal track name)
Uncompressed equivalent: ~526 bytes

Subsequent SUBSCRIBEs to Same Track

No encoder stream instructions needed; namespace elements are in the table, track name is sent as literal again:

```
 SUBSCRIBE Message:
   Type: 0x43
   Request ID: 2
   Track Alias: 101
   Compressed Block:
     Required Insert Count: 3
     Base: Sign=0, Delta=0
     Indexed Field Line (Dynamic, relative index 1)  // "conference"
     Indexed Field Line (Dynamic, relative index 0)  // "room42"
     Literal Field Line (Static Name 0x0C, Value "audio")  // TRACK_NAME
     Indexed Field Line (Dynamic, relative index 2)  // AUTH_TOKEN
```

Each subsequent SUBSCRIBE to the same track: ~12 bytes instead of ~526 bytes.

SUBSCRIBE to Different Track, Same Namespace

The track name "video" is also short, so we send it as a literal. No encoder stream instructions needed:

```
 SUBSCRIBE Message:
   Type: 0x43
   Request ID: 3
   Track Alias: 102
   Compressed Block:
     Required Insert Count: 3
     Base: Sign=0, Delta=0
     Indexed Field Line (Dynamic, relative index 1)  // "conference"
     Indexed Field Line (Dynamic, relative index 0)  // "room42"
     Literal Field Line (Static Name 0x0C, Value "video")  // TRACK_NAME
     Indexed Field Line (Dynamic, relative index 2)  // AUTH_TOKEN
```

The namespace elements are reused; the different track name is sent as a literal.

Code Point Summary

This appendix summarizes all code points defined or used by this extension.

Setup Options

Type	Name
0x10	MOQT_QPACK_MAX_TABLE_CAPACITY
0x11	MOQT_QPACK_BLOCKED_STREAMS
0x12	MOQT_QPACK_INDEX_SETUP_AUTH

Table 10

Stream Types

Type	Name
0x1f107a60	QPACK_ENCODER_STREAM
0x1f107a61	QPACK_DECODER_STREAM

Table 11

Pseudo-Parameter Types

Type	Name
0x0A	TRACK_NAMESPACE_ELEMENT
0x0B	TRACK_NAMESPACE_SET
0x0C	TRACK_NAME

Table 12

MOQPACK Message Types

The MOQPACK flag bit (0x40) is OR’d with standard MOQT message types:

Standard	MOQPACK	Message
0x02	0x42	REQUEST_UPDATE
0x03	0x43	SUBSCRIBE
0x04	0x44	SUBSCRIBE_OK
0x05	0x45	REQUEST_ERROR
0x06	0x46	PUBLISH_NAMESPACE
0x07	0x47	REQUEST_OK
0x08	0x48	NAMESPACE
0x0D	0x4D	TRACK_STATUS
0x0E	0x4E	NAMESPACE_DONE
0x11	0x51	SUBSCRIBE_NAMESPACE
0x16	0x56	FETCH
0x18	0x58	FETCH_OK
0x1D	0x5D	PUBLISH
0x1E	0x5E	PUBLISH_OK

Table 13

QPACK Library Adaptation Notes

This appendix summarizes the modifications needed to use a standard QPACK library (designed for HTTP/3) with this extension.

Static Table

Standard QPACK libraries include a static table of 99 predefined HTTP header (name, value) pairs. For MOQPACK, the static table is reinterpreted: the static table index directly represents the MOQT parameter type integer, and there are no predefined values. Implementations need to replace or bypass the HTTP static table. A simple approach is to treat the static table as a mapping from index to a 4-byte big-endian representation of the parameter type, with no

predefined value.

Prohibited Encodings

Standard QPACK supports all field line representations. MOQPACK prohibits several (see Section 6.3.1). Implementations should configure the encoder to only emit:

* Literal Field Line With Static Name Reference (for new parameter values)

* Indexed Field Line with Dynamic Table (for previously-inserted values)

* Indexed Field Line with Post-Base Index

And should configure the decoder to reject:

* Indexed Field Line with Static Table

* Literal Field Line With Dynamic Name Reference

* Literal Field Line with Post-Base Name Reference

* Literal Field Line With Literal Name

* Huffman-encoded string literals (H=1)

Encoder Stream Instructions

Only two insertion forms are used:

* Insert With Static Name Reference

* Duplicate

Insert With Dynamic Name Reference and Insert With Literal Name are not used.

Decoder Stream: Request IDs Instead of Stream IDs

Standard QPACK decoder instructions (Section Acknowledgment, Stream Cancellation) carry QUIC stream IDs. In MOQPACK, these carry MOQT Request IDs instead (see Section 7.2.1). The encoder needs to track Compressed Blocks per Request ID rather than per stream ID. This also enables MOQPACK to operate over WebTransport where QUIC stream IDs are not exposed to the application.

Entry Size Calculation

The name size for all entries is fixed at 4 bytes (rather than the variable-length header name strings used in HTTP). The 32-byte per-entry overhead from [QPACK] Section 3.2.1 still applies.

Author’s Address

Alan Frindell
Meta
Email: afrind@meta.com