Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 3 September 2026

D. Liu
Alibaba Cloud
S. Krishnan
Cisco

March 2026

Agent Protocol over MoQ
draft-liu-agent-protocol-over-moq-00

Abstract

This document specifies a Agent-to-Agent communication framework enabling structured, low-latency, and semantically rich communication between autonomous agents over the Media over QUIC (MoQ) protocol. It leverages MoQ’s efficient media transport capabilities while introducing a new application-layer framing mechanism to support control signaling, session management, and large data fragmentation. The design supports both intra-domain and inter-domain deployment, with an emphasis on interoperability, extensibility, and minimal overhead.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 2 September 2026.

Copyright Notice

Table of Contents

1. Introduction

The multimodal capabilities of AI Agents, has created a need for a real-time communication framework that supports both media streaming and structured control signaling. Existing protocols such as HTTP/2, gRPC, either suffer from head-of-line blocking or lack native support for real-time media and bidirectional streaming.

Media over QUIC (MoQ) offers a compelling transport foundation with its low-latency, multiplexed, and connection-robust design. However, MoQ is originally optimized for media delivery and lacks native semantics for agent-to-agent control, session establishment, and structured data exchange.

This document proposes an application-layer protocol framework that operates over MoQ. It introduces:

* A binary framing structure to carry both control and media payloads.

* A session management mechanism for multi-turn agent dialogues.

* Strategies for fragmenting large control signals with partial reliability.

* Support for both MoQ domain-local and internet-wide communication.

The proposed framework is designed to be minimal, modular, and compatible with existing MoQ implementations. It does not modify MoQ's transport semantics but instead defines a new application-layer envelope that can be carried within MoQ Objects or Datagrams.

2. Requirements and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

MoQ: Media over QUIC, a protocol for real-time media delivery.

Agent: An autonomous entity capable of sending and receiving structured commands and media.

Session: A logical dialogue between two agents, identified by a Session ID.

Frame: The basic unit of agent to agent communication, consisting of a header and payload.

3. Architecture Overview

Agent Protocol over MoQ operates as an application-layer protocol on top of MoQ. The protocol stack is illustrated below:

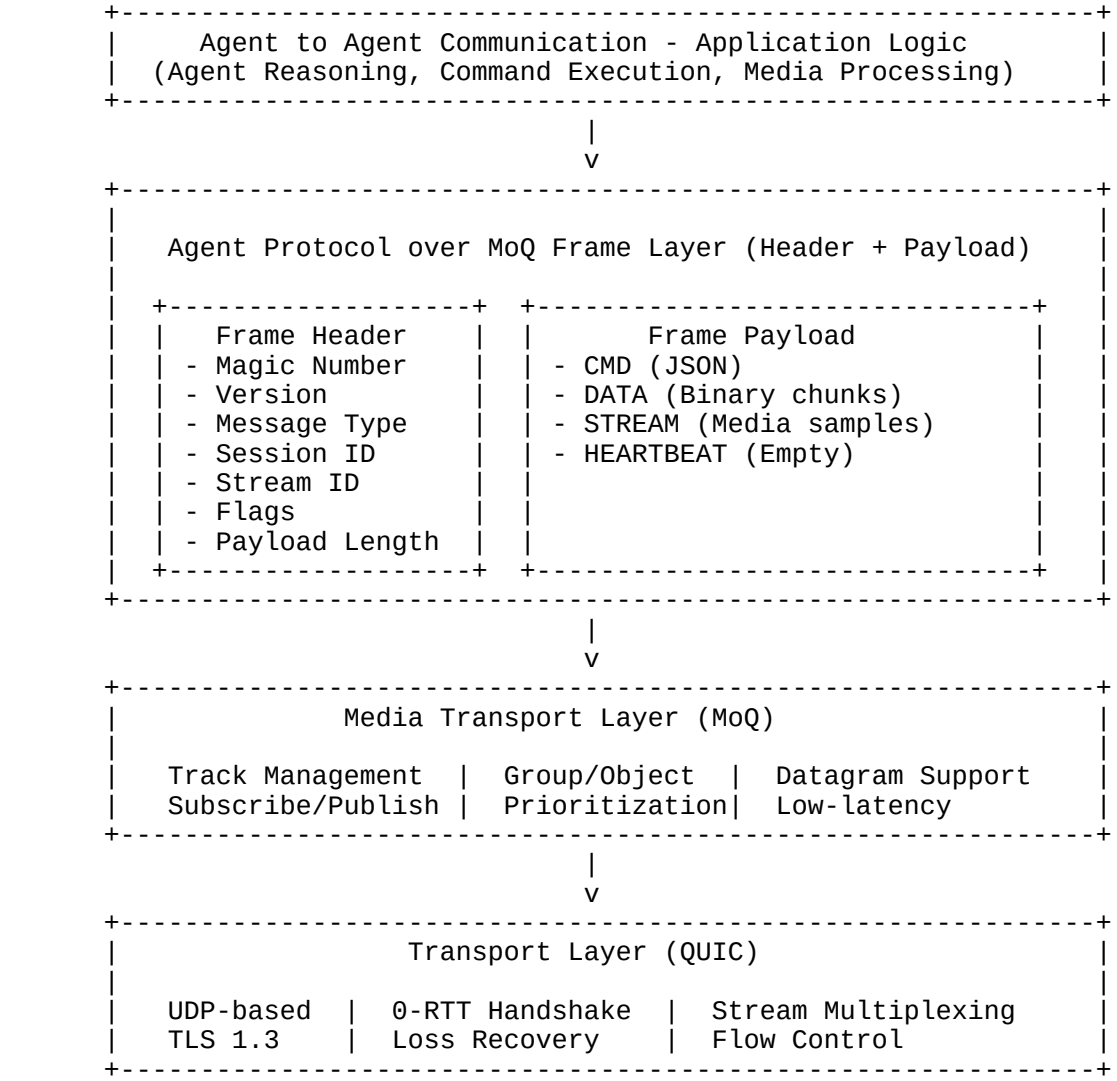

Figure 1: Protocol Stack of Agent Protocol over MoQ

The stack consists of four main layers:

* Transport Layer (QUIC): Provides UDP-based, low-latency transport with built-in encryption (TLS 1.3), 0-RTT connection establishment, stream multiplexing, and congestion control.

* Media Transport Layer (MoQ): Handles track-based pub/sub, group and object organization, priority management, and datagram support for real-time media delivery.

* Application Framing Layer: Defines the Agent Protocol over MoQ binary frame structure with fixed header fields and variable payload, enabling efficient encapsulation of different message types.

* Application Logic: Implements agent-specific functionality including reasoning engines, command interpretation, media processing, and business logic.

Agents establish a MoQ session (via WebTransport or native QUIC client) and use one or more Tracks for communication. Each agent-to-agent message is encapsulated in a binary frame before being sent as a MoQ Object or Datagram. The framing layer provides session management, fragmentation support, and extensibility through version and flag fields.

4. Agent to Agent Communication Frame Structure

The Agent to Agent Communication frame is a binary structure designed for efficiency and extensibility. It consists of a fixed-length header and a variable-length payload.

4.1. Frame Header Format

Field	Length (bytes)	Description	MoQ Mapping
Magic Number	2	Identifier: 0xA2A2	Detects agent-to-agent traffic
Version	1	Protocol version (e.g., 0x01)	Forward compatibility
Msg Type	1	Message type: CMD(0x01), DATA(0x02), STREAM(0x03), HEARTBEAT(0x04)	Route to handler
Session ID	8	Unique session identifier (UUIDv7 or hash)	Correlate multi-turn dialogues
Stream	4	Identifier for fragmented	Reassemble

ID		streams	large data
Flags	1	Bitmask: compressed(0x01), encrypted(0x02), fragmented(0x04)	Processing hints
Payload Len	4	Length of the following payload	Parse boundary

Table 1

4.2. Frame Payload

The payload is interpreted based on the Msg Type and Flags:

* CMD: JSON or MessagePack-encoded command structure.

* DATA: Raw binary chunk of a large file or buffer.

* STREAM: Raw media sample or simplified RTP-like header + frame.

* HEARTBEAT: Empty payload; used for liveness detection.

4.3. Message Encoding Formats

Agent Protocol over MoQ uses JSON (JavaScript Object Notation) as the standard encoding format for structured message content:

4.3.1. JSON Encoding

JSON (JavaScript Object Notation) is used for all CMD and control messages:

* *Human-readable* and widely supported across programming languages

* *Self-describing* with key-value pairs

* *Schema-flexible* allowing dynamic field addition

* *Standard library support* in most modern languages

Example CMD message in JSON:

```
{
  "action": "session_init",
  "session_id": "abc-123-def-456",
  "auth_token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
  "capabilities": {
    "media_types": ["video/h264", "audio/opus"],
    "max_fragment_size": 1400
  }
}
```

Figure 2

4.3.2. Binary Encoding for STREAM and DATA

Media streams and large data payloads use raw binary encoding:

* *STREAM messages*: Raw media samples (H.264, Opus, etc.) without additional encoding overhead

* *DATA messages*: Raw binary chunks for file transfers and large payloads

* *Maximum efficiency* for bandwidth-constrained scenarios

4.4. Message Type Details

Each message type has specific field requirements and processing rules:

4.4.1. CMD Message Format

CMD messages carry structured commands encoded in JSON. In addition to session management commands, CMD messages include specific actions for subscription management in pub/sub scenarios.

Field	Type	Required	Description	Example
action	String	Yes	Command identifier	"session_init", "session_ack", "subscribe_request", "unsubscribe_request", "track_ready"
session_id	String/UUID	Yes (for session cmds)	Session identifier	"abc-123-def-456"
auth_token	String	Yes (for session_init)	JWT with operation claim	"eyJhbGciOiJIUzI1NiIsIn..."
status	String	No	Response status	"approved", "denied", "error"
error_code	Integer	No	Error identifier	401, 403, 500
payload	Any	No	Command-specific data	{"key": "value"}

Table 2

4.4.2. Subscription Management Fields

For subscription-related actions such as "subscribe_request" and "unsubscribe_request", CMD messages include additional fields:

Field	Type	Required	Description	Example
track_name	String	Yes (for subscribe)	Name of the track to subscribe to	"agent2.session456.video"
quality_params	Object	No	Quality parameters for the subscription	{"resolution": "1080p", "fps": 30}
subscriber_id	String	Yes (for subscribe)	ID of the subscribing agent	"agent1"

Table 3

4.4.3. DATA Message Format

DATA messages carry raw binary chunks for large file transfers.

Field	Type	Required	Description	Constraints
chunk_index	Integer	Yes	Position in sequence	0-based, sequential
total_chunks	Integer	Yes (last chunk)	Total number of chunks	Must match in all chunks
checksum	String	No	Data integrity hash	SHA-256 hex encoded
metadata	Object	No	File/type information	{"filename": "file.txt"}

Table 4

4.4.4. STREAM Message Format

STREAM messages carry media samples with optional timing information.

Field	Type	Required	Description	Example
timestamp	Integer	Yes	Sample timestamp (ms)	1678901234567
media_type	String	Yes	Media content type	"video/h264", "audio/opus"
sequence_number	Integer	Yes	Frame sequence	12345
key_frame	Boolean	No	Is key frame	true
bitrate	Integer	No	Current bitrate (bps)	1000000

Table 5

4.4.5. HEARTBEAT Message Format

HEARTBEAT messages have empty payload and are used for liveness detection.

Field	Value	Description
Payload Length	0	No payload data
Processing	Ack only	Recipient should acknowledge receipt

Table 6

4.5. MoQ Mapping and Transport Considerations

Agent Protocol over MoQ frames are mapped to MoQ Objects and Datagrams based on their reliability and latency requirements:

* *CMD frames*: Mapped to MoQ Objects to ensure reliable delivery of control messages. These require acknowledgment and retransmission if lost.

* *STREAM frames*: Mapped to MoQ Objects by default for ordered delivery. May use MoQ Datagrams for loss-tolerant real-time streams where occasional packet loss is acceptable.

* *DATA frames*: Large data chunks are mapped to MoQ Objects for reliable transfer. Small data may use MoQ Datagrams for lower latency.

* *HEARTBEAT frames*: Mapped to MoQ Datagrams for efficient keep-alive functionality without requiring acknowledgments.

This mapping ensures that the transport characteristics of MoQ align with the application requirements of agent-to-agent communication.

4.6. Pub/Sub Mechanism Integration

The MoQ pub/sub mechanism is integrated with Agent Protocol over MoQ through specific CMD message types that manage subscriptions at the application layer:

* *Subscription Requests*: Agents send CMD messages with action "subscribe_request" to initiate track subscriptions.

* *Subscription Management*: The "subscribe_request" includes track name, quality parameters, and authorization tokens.

* *Track Naming Convention*: Track names follow the format "{agent_id}.{session_id}.{stream_type}", e.g., "agent1.abc-123-video".

* *Authorization*: Subscription requests are validated against the agent’s authorization token before establishing the underlying MoQ subscription.

This approach provides application-layer control over the underlying MoQ pub/sub mechanism, allowing fine-grained access control and session management.

5. Session and Communication Model

Agent Protocol over MoQ supports two communication modes:

5.1. In-Session Communication (Recommended)

Agents establish a session using a handshake that includes authorization verification. Once a session is established, agents can use subscription management commands to establish pub/sub relationships:

* Agent A obtains an authorization JWT containing an operation claim as defined in [I-D.liu-agent-operation-authorization].

* Agent A sends a CMD frame with {action: "session_init", session_id: 0x123..., auth_token: "<JWT>"}.

* Agent B validates the JWT and verifies the operation permissions before responding.

* Agent B responds with CMD {action: "session_ack", session_id: 0x123..., status: "approved|denied"}.

* Subsequent messages use the same Session ID in the frame header.

The session terminates with a *session_bye* command or MoQ session closure.

5.2. Session-Less Communication

For stateless interactions (e.g., heartbeat, fire-and-forget command), agents may send frames without session setup. The *Session ID* field is set to zero. Authorization for such interactions is typically pre-established or implicit.

6. Fragmentation and Large Data Handling

Agent Protocol over MoQ defines three strategies based on payload size:

6.1. Small Signals (< 1400 bytes)

Sent as a single MoQ Datagram or Object. No application-layer fragmentation. Ideal for commands and heartbeats.

6.2. Large Data (> 1400 bytes)

Application-layer fragmentation is required:

* Data is split into chunks (e.g., 1400 bytes).

* Each chunk is encapsulated in an agent protocol over MoQ frame with the same Stream ID.

* The last chunk sets the fragmented flag to 0 and includes final metadata.

Reassembly is performed at the receiver using Stream ID and order of arrival.

6.3. Streaming Media

Continuous media (audio/video) is sent as a sequence of agent protocol over MoQ *STREAM* frames. Each frame carries one media sample. Loss of a single frame does not block subsequent ones, leveraging MoQ's partial reliability.

7. Transport and Domain Considerations

Agent Protocol over MoQ is designed to operate in two modes:

* Intra-MoQ-Domain: Agents communicate within the same MoQ infrastructure (e.g., same edge cluster). Low latency, high trust. MoQ-native discovery is RECOMMENDED.

* Inter-Domain (Internet-wide): Agents communicate across networks. STUN/TURN/ICE MAY be used if direct QUIC path fails.

Agent Protocol over MoQ does not define its own discovery mechanism but RECOMMENDS integration with MoQ's track discovery or external service directories.

8. Security Considerations

Agent Protocol over MoQ inherits security from the underlying transport:

* QUIC provides TLS 1.3 encryption and 0-RTT resumption.

* Application-layer authorization uses JWT with operation claim as defined in [I-D.liu-agent-operation-authorization].

* Session establishment requires validation of authorization tokens before communication begins.

Implementations MUST validate authorization tokens before processing any agent-to-agent communication. Session IDs MUST be cryptographically random to prevent session hijacking.

9. IANA Considerations

This document requests IANA to create new registries and register initial values for the Agent Protocol over MoQ.

9.1. Agent Protocol over MoQ Magic Number Registry

IANA is requested to create a new registry for Agent Protocol over MoQ Magic Numbers under the "Media over QUIC (MoQ)" group. The registration procedure is Standards Action as defined in [RFC8126].

Initial registration:

Value	Description	Reference
0xA2A2	Agent-to-Agent Communication Frame	[This document]

Table 7

9.2. Agent Protocol over MoQ Version Registry

IANA is requested to create a new registry for Agent Protocol over MoQ Versions. The registration procedure is Standards Action.

Initial registration:

Value	Description	Reference
0x01	Agent Protocol over MoQ Version 1	[This document]

Table 8

Values 0x00 and 0xFF are reserved. Values 0x02-0xFE are available for assignment.

9.3. Agent Protocol over MoQ Message Type Registry

IANA is requested to create a new registry for Agent Protocol over MoQ Message Types. The registration procedure is Specification Required as defined in [RFC8126].

Initial registrations:

Value	Message Type	Description	Reference
0x00	Reserved	Reserved for future use	[This document]
0x01	CMD	Command message with structured payload	[This document]
0x02	DATA	Raw binary data chunk	[This document]
0x03	STREAM	Media stream sample	[This document]
0x04	HEARTBEAT	Liveness detection message	[This document]

Table 9

Values 0x05-0xFE are available for assignment. Value 0xFF is reserved.

9.4. Agent Protocol over MoQ Frame Flags Registry

IANA is requested to create a new registry for Agent Protocol over MoQ Frame Flags. The registration procedure is Specification Required. Each flag is represented by a bit position (0-7).

Initial registrations:

Bit	Flag Name	Description	Reference
0	Compressed	Payload is compressed (0x01)	[This document]
1	Encrypted	Payload has application-layer encryption (0x02)	[This document]
2	Fragmented	Frame is part of a fragmented stream (0x04)	[This document]
3-7	Unassigned	Available for assignment	[This document]

Table 10

9.5. Well-Known URI Registration

IANA is requested to register a new well-known URI in the "Well-Known URIs" registry as defined in [RFC8615].

The "/.well-known/a2a" endpoint facilitates agent discovery in inter-domain deployments. When accessed via HTTPS GET, the endpoint returns a JSON response containing the MoQ endpoint information for Agent Protocol over MoQ communication.

URI suffix: a2a

Change controller: IETF

Specification document: This document

Related information: Used for WebTransport-based Agent Protocol over MoQ endpoint discovery. The endpoint returns a JSON object with the MoQ endpoint information, e.g., { "moq_endpoint": "quic://api.example.com:443" }

10. Normative References

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

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

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

11. Informative References

[MOQ] Nisancioglu, M. D., Enghardt, T., and C. Perkins, "Media over QUIC (MoQ) - Internet Draft", I-D ietf-moq-00, 2024.

[WEBTRANS] Jansson, P. and J. Uberti, "WebTransport over HTTP/3", RFC 9220, August 2022, <https://www.rfc-editor.org/rfc/rfc9220>.

[I-D.liu-agent-operation-authorization]
Liu, D., Zhu, H., and S. Krishnan, "Agent Operation Authorization Policy", I-D liu-agent-operation-authorization, January 2026.

Appendix A. Appendix A: Usage Examples

This appendix provides detailed examples of Agent Protocol over MoQ interactions based on a multimodal collaborative design scenario.

A.1. A.1 Scenario Overview

Agents:

* *Agent 1 (Personal Assistant):* Resides on a user’s smartphone.

* *Agent 2 (Professional Designer):* Resides on a cloud server.

A.2. A.2 Stateless Communication (Session-Less)

Used for lightweight, fire-and-forget interactions like liveness checks or simple status updates.

A.2.1. A.2.1 Heartbeat Interaction

Agent 1 sends a heartbeat to Agent 2 to maintain the MoQ track activity.

Header:
- Magic Number: 0xA2A2
- Version: 0x01
- Msg Type: 0x04 (HEARTBEAT)
- Session ID: 0 (Stateless)
- Stream ID: 0
- Flags: 0x00
- Payload Len: 0
Payload: (Empty)

Figure 3: Heartbeat Frame Implementation

A.3. A.3 Stateful Communication with Streaming Media

Used for complex dialogues requiring continuous media processing, such as real-time video analysis.

A.3.1. A.3.1 Real-time Video Stream for Design Feedback

Step 1: Session Handshake with Authorization

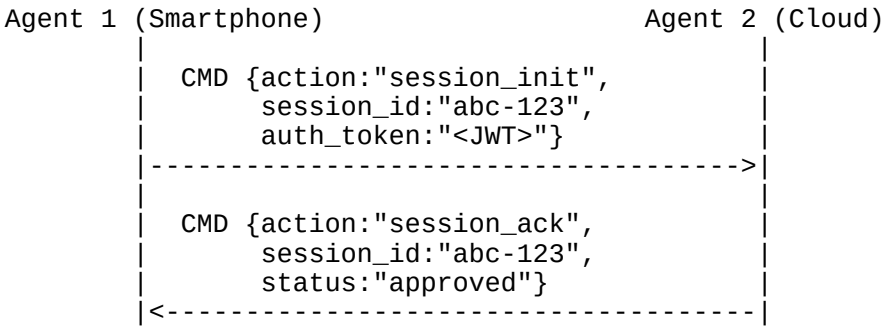

Figure 4: Session Establishment Sequence

Step 2: Media Streaming

Agent 1 captures video and sends it as STREAM frames within the established session.

Header:
- Magic Number: 0xA2A2
- Version: 0x01
- Msg Type: 0x03 (STREAM)
- Session ID: abc-123
- Flags: 0x00
- Payload Len: 1200

Payload: [Raw H.264/HEVC NAL Unit]

Figure 5: Video Sample Frame Implementation

A.3.2. A.3.2 Subscription Management Example

Step 1: Subscription Request

Agent 1 wants to subscribe to Agent 2’s video feed. It sends a subscription request within the established session.

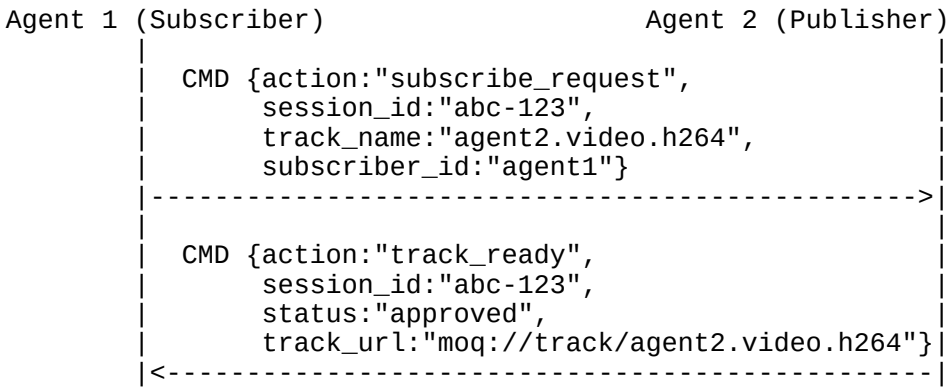

Figure 6: Subscription Request Sequence

Step 2: Media Streaming via MoQ pub/sub

After successful subscription negotiation via CMD messages, the actual media stream flows through the underlying MoQ pub/sub mechanism.

A.4. A.4 Stateful Communication with Large Data Blocks

Used for transferring high-resolution assets or large model weights.

A.4.1. A.4.1 Sending High-Res Design Assets

Agent 2 sends a 5MB design file back to Agent 1.

Fragmentation Strategy: The file is split into chunks of 1400 bytes.

Header:
- Msg Type: 0x02 (DATA)
- Session ID: abc-123
- Stream ID: 101 (File Transfer Stream)
- Flags: 0x04 (Fragmented = 1)
- Payload Len: 1400
Payload: [First 1400 bytes of the file]

Figure 7: First Chunk Frame

```
Header:
- Msg Type: 0x02 (DATA)
- Session ID: abc-123
- Stream ID: 101
- Flags: 0x00 (Fragmented = 0)
- Payload Len: 800
Payload: [Final 800 bytes of the file]
```

Figure 8: Last Chunk Frame

A.5. A.5 Deployment Scenarios

A.5.1. A.5.1 Intra-Domain (Same Edge Cluster)

Agents are deployed in the same regional data center (e.g., Alibaba Cloud Hangzhou Region).

* *Discovery:* Uses MoQ-native track discovery.

* *Latency:* Extremely low (< 5ms).

* *Trust:* High; encryption is present but overhead is minimal due to short physical distance.

A.5.2. A.5.2 Inter-Domain (Cross-Cloud Invocation)

Agent 1 (on Smartphone) calls Agent 2 (on Cloud).

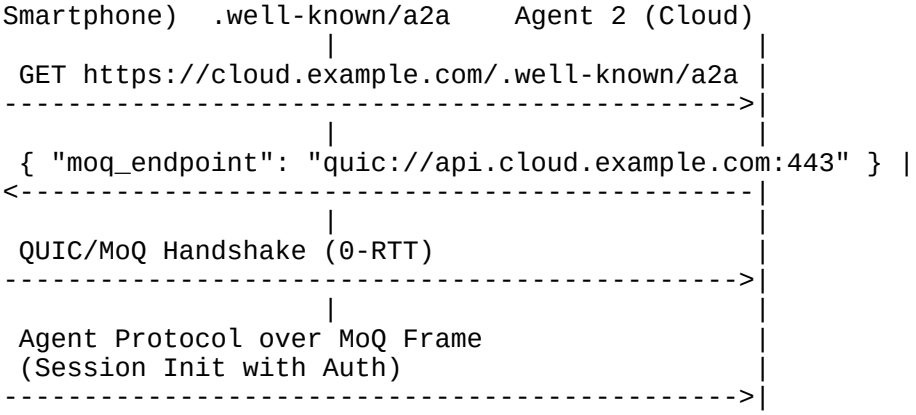

Figure 9: Inter-Domain Connection Flow

* *Discovery:* Uses HTTPS Well-Known URI as a bootstrap.

* *Traversal:* Relies on QUIC’s robustness against NAT changes.

* *Security:* Inherits TLS 1.3 from QUIC; ensures end-to-end privacy across public networks.

A.6. A.6 Summary Table of Interaction Types

Type	Session ID	Msg Type	Typical Usage	MoQ Mapping
Session-Less	0	HEARTBEAT / CMD	Keep-alive, Simple Queries	MoQ Datagram (HEARTBEAT), MoQ Object (CMD)
Streaming	UUID	STREAM	Audio/Video/Sensor Stream	MoQ Object (Ordered Delivery), MoQ Datagram (Loss-Tolerant)
Bulk Data	UUID	DATA	Large Files, Images	MoQ Object (Reliable), MoQ Datagram (Small Data)
Control	UUID	CMD	Logic Execution, ACKs, Subscription Management	MoQ Object (Reliable)

Table 11

Authors' Addresses

Dapeng Liu
Alibaba Cloud
Email: max.ldp@alibaba-inc.com

Suresh Krishnan
Cisco
Email: suresh.krishnan@gmail.com