Technical Note

An HTTP/3 CDN-Based Object-Centric Architecture for Real-Time Media Delivery

Daisuke Sugisawa[1,a)]

Abstract: In recent years, low-latency real-time media delivery technologies represented by WebRTC and WebTransport have become widespread. However, many of these technologies assume a session-oriented communication model, and face challenges in resilience to frequent disconnections and reconnections in mobile environments, as well as cost efficiency during large-scale delivery. On the other hand, while CDNs provide high scalability and stable transfer performance, they have traditionally been considered unsuitable for real-time applications. This study redefines real-time media delivery not as session-oriented communication, but as a delivery problem of continuously updated objects. In the proposed method, the latest media data is delivered as objects using long-lived connections over HTTP/3, and state management is decoupled from the transport layer. As a result, even when disconnections occur, delivery can be immediately resumed through offset-based retrieval without requiring reconstruction of session state. Furthermore, by assuming delivery through CDNs, the proposed method achieves a linear cost model proportional to transfer volume even as the number of end users increases. In evaluation experiments, Time To First Audio Byte (TTFAB) and operational costs were measured and compared with conventional WebRTC and SFU configurations. The results showed that the proposed method achieved approximately 3x faster TTFAB (0-RTT, 180ms) compared to WebRTC P2P (best case, 500 to 600ms), with an order of magnitude difference when candidate fallback occurs in NAT environments. Advantages in cost efficiency during large-scale delivery were also confirmed. This study presents a practical architecture for real-time media delivery through object-centric design using HTTP/3 CDN.

Keywords: CDN, HTTP/3, QUIC, Real-Time Media Delivery, Object-Centric, Low-Latency Delivery

1. Introduction

In recent years, low-latency real-time media delivery technologies represented by WebRTC and WebTransport have become widely used. These technologies adopt designs that assume transport-level session management to achieve bidirectional communication and low latency, and play important roles in online conferencing and interactive streaming services. However, since these methods manage media state tightly coupled to transport sessions, they are vulnerable to temporary disconnections and reconnections in mobile environments, and face the challenge of requiring state reconstruction and resynchronization when disconnections occur.

Furthermore, in large-scale delivery using WebRTC, it is necessary to deploy many relay nodes such as SFUs (Selective Forwarding Units), and it has been pointed out that infrastructure costs and operational burden increase non-linearly as the number of connections and concurrent viewers grows. Against this background, there is a demand for real-time media delivery methods that maintain low latency while achieving both high reconnection resilience and scalability.

On the other hand, CDNs (Content Delivery Networks) are widely used as infrastructure that provides high scalability and stable transfer performance for large-scale object delivery. However, conventional CDNs have been designed primarily for file delivery and on-demand streaming, and have been considered unsuitable for real-time applications, especially delivery requiring latency on the order of tens of milliseconds. Therefore, designs that actively utilize CDNs in real-time media delivery have not been sufficiently explored until now.

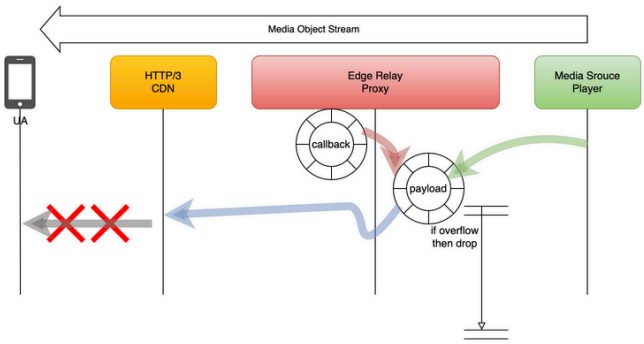

Fig. 1: Object-Centric Real-Time Media Delivery Architecture

In contrast to these conventional assumptions, this study redefines real-time media delivery not as session-oriented communication, but as a delivery problem of continuously updated objects (Figure 1). In the proposed method, media data is delivered as objects using long-lived connections over HTTP/3, and media state is managed separately from transport sessions. As a result, even when disconnections occur, delivery can be immediately resumed through offset-based retrieval of objects without requiring

[1] Xander, LLC. Shibuya, Tokyo, Japan
[a)] daisuke.sugisawa.xander@gmail.com

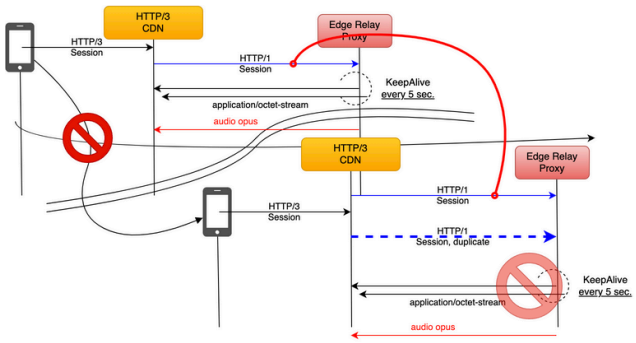

Fig. 2: CDN Session Control

reconstruction of session state.

Furthermore, by assuming delivery through CDNs, the proposed method achieves a linear cost model proportional to transfer volume even as the number of end users increases (Figure 2). This characteristic enables suppression of infrastructure complexity and rapid increase in operational costs even in real-time delivery with large-scale concurrent viewing.

In this paper, we first organize the challenges in existing session-oriented methods and present the hypotheses of this study. Next, we present the object-centric real-time media delivery architecture using HTTP/3 CDN as the proposed method, and describe its design principles and characteristics. Furthermore, we verify the effectiveness of the proposed method through comparative experiments with conventional methods, and finally present a summary and future challenges.

2. Challenges

This chapter organizes the challenges faced by existing real-time media delivery technologies.

2.1 Limitations of Session-Oriented Communication

Existing real-time communication technologies represented by WebRTC and WebTransport adopt designs that tightly couple media state management to transport sessions. This design has the following fundamental challenges:

- **Lack of reconnection resilience**: Session state is lost during network disconnections, and reconnection requires state reconstruction such as ICE renegotiation and DTLS handshake. Disconnections occur frequently in mobile environments, leading to degradation of user experience.
- **Complexity of state management**: Playback position, buffer state, codec parameters, etc. are tied to sessions, making resynchronization processing after disconnection complex.
- **Scalability constraints**: Since each session maintains independent state, session management overhead on the server side is large, leading to non-linear resource consumption during large-scale delivery.

2.2 Cost Structure in Large-Scale Delivery

In large-scale delivery using WebRTC, architectures centered on SFUs (Selective Forwarding Units) are common. However, this configuration has the following cost challenges:

- **Non-linear cost increase**: As the number of viewers increases, SFU instances need to be increased, causing costs to increase non-linearly.
- **Operational complexity**: Operational burden is high for SFU cluster load balancing, failover, geographic distribution, etc.
- **Incompatibility with CDN**: Caching at stateless CDN edges is difficult because session state needs to be maintained.

2.3 Limitations of Existing Low-Latency Streaming

HTTP-based streaming such as HLS and DASH is said to have high affinity with CDNs, but has the following fundamental challenges for real-time applications:

- **Segment-based latency**: Latency of at least several hundred milliseconds to several seconds occurs.
- **Increased origin load**: Shortening segment length for low latency increases the frequency of segment generation, encoding, and file writing at the origin server, causing CPU and I/O load to increase sharply. LL-HLS uses partial segments, but this requires even finer-grained object management, worsening origin load.
- **CDN distribution cost explosion**: Finely divided segment files are distributed to CDN edges, incurring request charges and cache invalidation costs proportional to the number of objects. For 1-second segments over 1 hour of delivery, 3,600 objects need to be managed and delivered.
- **Manifest update overhead**: Playlists (m3u8/mpd) need to be updated and delivered at high frequency, and polling requests from clients concentrate load on origin and CDN edges.
- **Lack of bidirectionality**: One-way delivery is assumed, and interactive applications cannot be supported.

These challenges stem from the segment-based design of "delivering media as groups of fragmented objects." The proposed method fundamentally eliminates these overheads by treating media as a "single continuous object."

3. Design Rationale

Based on the challenges organized in the previous chapter, this study presents the following design principles.

3.1 State Management Separation

Design Principle 1: By separating media state from transport sessions and managing it as continuously updated objects, session-independent reconnection becomes possible, improving reconnection resilience in mobile environments.

This principle overturns the conventional assumption of "state = session" in session-oriented models and proposes a new model of "state = object offset."

3.2 CDN Utilization

Design Principle 2: By combining long-lived connections over HTTP/3 with CDN edge caching, CDN transfer volume billing models can be utilized even in real-time delivery, achieving linear cost scaling with respect to the number of viewers.

This principle challenges the conventional wisdom that "CDNs are unsuitable for real-time delivery."

3.3 Evaluation Metrics

To verify the above hypotheses, the following items are evaluated:

- Time To First Audio Byte (TTFAB): Latency from HTTP request transmission to first audio payload reception
- Operational cost structure during large-scale delivery

Measuring "reconnection time" in conventional methods is inappropriate as a fair comparison metric because it depends on protocol-specific disconnection detection delays such as TCP timeout and ICE timeout. This study quantitatively evaluates the "no session reconstruction required" characteristic of the proposed method by measuring the time from when the UA initiates a new connection to when it actually receives media data (TTFAB).

4. Proposed Method

This chapter presents the object-centric real-time media delivery architecture based on the hypotheses in the previous chapter.

4.1 Design Principles

The core of the proposed method is redefining real-time media delivery not as "session-oriented communication" but as "object delivery." Specifically, it is based on the following design principles:

- **Media as objects**: Real-time video and audio are treated as single objects that are continuously updated. The current position of media is expressed as a 64-bit extended sequence number.
- **Externalization of state**: Media state is separated from transport sessions and managed as 64-bit extended sequence numbers of RTP packets.
- **CDN-native design**: Long-lived connections over HTTP/3 are used, assuming caching and delivery at CDN edges.

In the proposed method, real-time video is treated as continuously updated objects and delivered through HTTP/3 CDN. Media state is managed as object offsets and is decoupled from transport sessions, so state reconstruction is not required even when disconnections occur.

4.2 Difference from Traditional HTTP Range Requests

The proposed method is superficially similar to traditional Range requests for static files in that it retrieves data from a specified offset over HTTP. However, it differs fundamentally in the following aspects:

- **Nature of target object**: Traditional Range requests target completed files with a fixed Content-Length. The proposed method targets a "growing single object" that is continuously appended in real-time, receiving the latest data continuously like `tail -f`.
- **CDN affinity**: Traditional Range requests have poor partial cache efficiency and are unsuitable for live use cases. The proposed method is designed with CDN in mind, achieving high cache efficiency for mass delivery of identical objects.

In other words, the essence of this method is not "reading static files over HTTP," but "abstracting real-time media state itself as an HTTP object."

4.3 Protocol Stack Comparison

WebRTC and WebTransport maintain media state within transport sessions, so state reconstruction is required when disconnections occur. In contrast, the proposed method manages media state as continuously updated objects and achieves session-independent reconnection through offset-based retrieval over HTTP/3. The essential difference lies not in communication latency, but in where the media state is maintained.

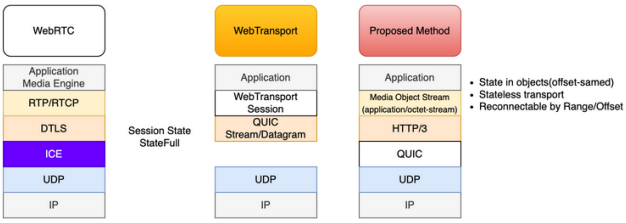

Fig. 3: Protocol Stack Comparison

4.4 Protocol Architecture

The protocol architecture of the proposed method is shown below.

(1) **UA → CDN Edge**: Connect to CDN edge via HTTP/3 over QUIC. QUIC 0-RTT reconnection minimizes handshake overhead.

(2) **CDN Edge → Edge Relay Proxy**: CDN edge connects to Edge Relay Proxy (Origin) as HTTP/1.1 TCP Upstream.

(3) **HTTP Request Headers**: Requests from UA include the following custom headers. CDN edge transparently forwards these headers to Edge Relay Proxy.

- `X-Publish-ID`: Unique ID identifying the delivery stream
- `X-Proxy-Offset`: 64-bit extended sequence number to start receiving from

(4) **Stream Return**: Edge Relay Proxy identifies the stream specified by `X-Publish-ID` and returns an octet stream as `application/octet-stream` from the `X-Proxy-Offset` position. CDN edge forwards this stream to UA.

Edge Relay Proxy is deployed in each region and holds and delivers media streams. CDN edge handles caching and delivery, avoiding load concentration on Edge Relay Proxy and achieving scalable delivery.

4.4.1 Difference from Conventional CDN Behavior

In conventional CDNs, the edge initiates a connection to the origin and fetches content upon arrival of the first client request (origin fetch). While this on-demand behavior is suitable for static content delivery, it introduces origin fetch latency for the first viewer in real-time delivery scenarios.

In contrast, the proposed architecture maintains a continuously open upstream stream (HTTP/1.1 TCP Upstream) from the Edge Relay Proxy to CDN edges. That is, the stream from Edge Relay Proxy to CDN edge is already established at the time delivery begins, and CDN edges do not need to initiate a new origin connection upon the arrival of the first subscriber. As a result, no

origin fetch latency is incurred even for the first viewer, and CDN edges can immediately deliver the already-arriving stream data.

4.5 Reconnection Mechanism

Reconnection in the proposed method does not require reconstruction of session state. UA simply issues a new HTTP request and specifies the last received position with `X-Proxy-Offset`. This mechanism eliminates the need for ICE renegotiation and DTLS handshake required in conventional WebRTC, achieving significantly reduced reconnection time.

4.5.1 Burst Delivery and Re-synchronization

Edge Relay Proxy caches received RTP packets in a ring buffer. When the 64-bit extended sequence number specified by `X-Proxy-Offset` in a reconnection request from UA exists in the ring buffer, Edge Relay Proxy sends all cached packets from that sequence number onwards in bulk (Burst Delivery).

Through Burst Delivery, UA rapidly receives packets accumulated during disconnection and catches up to the current stream position, completing Re-synchronization. This method has the following characteristics:

- **L2 cache locality optimization**: Since the ring buffer is placed in contiguous memory regions, temporal and spatial locality of L2 cache is maximized during Burst Delivery, improving transfer efficiency
- **Minimized playback interruption**: When disconnection period is short and the relevant sequence number remains in the buffer, UA can continue stream without packet loss
- **Graceful degradation**: When the relevant sequence number is lost from the buffer, it falls back to delivery from the current position, recovering with partial playback interruption

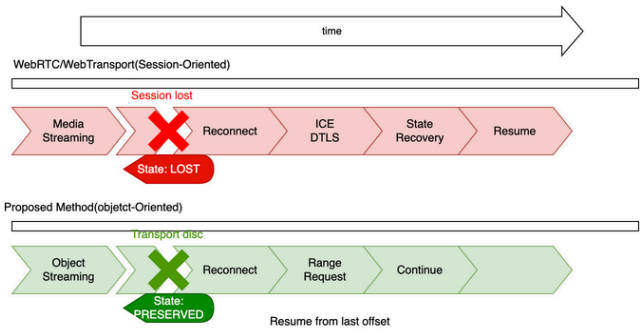

Fig. 4: Comparison of Reconnection Behavior

4.6 Object Format

In the proposed method, media data is objectified in the following format:

- **Codec**: OPUS (audio), VP8/VP9/AV1 (video)
- **Packetization**: PES (Packetized Elementary Stream)
- **Container**: MPEGTS
- **Transfer**: HTTP/3 octet stream (application/octet-stream)

This format maintains compatibility with existing media processing pipelines while achieving efficient delivery over HTTP/3 CDN.

5. Related Work

5.1 Session-Oriented Real-Time Communication

WebRTC [1] is widely adopted as a standard technology for achieving low-latency bidirectional media communication between browsers. WebRTC combines media transfer using RTP/RTCP with session establishment and encryption through ICE and DTLS, enabling low-latency and reliable real-time communication. This design demonstrates high effectiveness in interactive communication and small-group conferencing applications.

On the other hand, since WebRTC adopts a design that tightly couples media state management to transport sessions, sessions are lost during network disconnections, and reconnection processing and state reconstruction are required to resume media delivery. This characteristic poses challenges from the perspective of resilience to temporary disconnections that frequently occur in mobile environments.

Furthermore, to achieve large-scale delivery, it is necessary to deploy many relay nodes such as SFUs, and there is a tendency for infrastructure configuration and operational costs to become complex and expensive as the number of viewers increases.

WebTransport has been proposed as a new bidirectional communication API based on QUIC [7], aiming to achieve low-latency data transfer over HTTP/3 [6]. WebTransport has a simpler protocol structure compared to WebRTC and is characterized by flexible use of streams and datagrams. However, WebTransport also assumes a session-oriented communication model, and since session state is lost during disconnections, resumption processing is left to application-level implementation.

5.2 HTTP-Based Low-Latency Streaming

As media delivery methods using HTTP, HLS [2] and MPEG-DASH [3] are widely used, and in recent years, extension methods to reduce latency such as CMAF [4] and Low Latency HLS (LL-HLS) [5] have been proposed. These methods have high affinity with CDNs and provide excellent scalability and stability in large-scale delivery.

However, these methods fundamentally assume segment-based delivery, and latency often ranges from hundreds of milliseconds to several seconds. Therefore, they have been considered unsuitable for real-time media delivery requiring bidirectionality and immediacy. Furthermore, efforts to reduce latency have focused mainly on segment size and playout control, and have not delved into redesigning the communication model itself.

5.3 QUIC-Native Media Delivery

Recently, the IETF established the Media over QUIC (MoQ) [8] working group, which is standardizing real-time media delivery protocols over QUIC. MoQ treats media as objects (Named Objects) rather than sessions, and realizes publish-subscribe delivery through relay nodes (MoQ Relay). As related protocols, WARP [9] proposes low-latency media delivery over WebTransport, and RUSH [10] designs reliable real-time delivery over QUIC streams.

These efforts share the fundamental concept of "treating media as objects" with this study, and are positioned in the same trend of departing from the session-oriented model. Table 1 shows a

comparison of key design elements.

Table 1: Design Comparison: MoQ vs Proposed Method

Design Element	MoQ	Proposed Method
Core Concept	Media as Named Object	Media as Continuous Object
Transport	QUIC-native	HTTP/3 over QUIC
Relay Node	MoQ Relay (new)	CDN Edge (existing)
Cache Unit	Object / Group	Packet / Stream
Standardization	IETF (in progress)	Implementation-driven

The essential difference is that MoQ assumes a new QUIC-native protocol stack and the construction of dedicated relay infrastructure, whereas the proposed method directly leverages HTTP/3 and existing CDN infrastructure. MoQ pursues a flexible object model and protocol generality, while the proposed method prioritizes immediate deployability on existing infrastructure and operational cost efficiency.

In other words, the proposed method can be positioned as an approach that proactively implements the vision of "real-time delivery treating media as objects" — which MoQ seeks to realize through standardization — on existing CDN infrastructure.

5.4 Positioning of This Research

In contrast to the above existing research, this study is characterized by reconsidering real-time media delivery not as a problem of session-oriented communication, but as a problem of delivering continuously updated objects. This direction is shared with the vision pursued by MoQ/WARP/RUSH, and this study independently arrived at the same design philosophy and demonstrated it on existing CDN infrastructure.

The proposed method does not deny the low latency provided by WebRTC or WebTransport, but presents a different design space from the session-centric state management model they have assumed. Furthermore, while MoQ advances standardization of a new protocol stack and dedicated relays, the proposed method demonstrates that the same design principles can be realized on the mature infrastructure of HTTP/3 and existing CDNs.

While leveraging the high affinity with CDNs that HTTP-based streaming methods possess, by combining long-lived connections over HTTP/3 with offset-based retrieval, we aim to achieve both reconnection resilience and low latency for real-time applications. In this way, this study is positioned as complementary to MoQ standardization efforts, and as an attempt to redefine the nature of state management in real-time media delivery.

6. Evaluation

The purpose of this study is to redefine the communication model in real-time media delivery and clarify the characteristics that this design brings. Therefore, in this chapter, evaluation is conducted not from the perspective of simply comparing latency performance superiority, but from the perspective of what **operational characteristics** the proposed method possesses.

6.1 Operational Cost Structure Analysis (Fully Managed)

The essence of the proposed method lies in achieving the low-latency real-time delivery that WebTransport aimed for by realizing edge-proximity caching through the Edge Relay Proxy deployed behind CDN edges. This enables the construction of HTTP/3-based real-time CDN in a fully managed configuration.

The operational cost estimation (fully managed) is performed based on the following assumptions:

- 1 user = 1 session/day
- Average viewing time: 30 minutes
- Concurrent connections: 10% of DAU
- Data rate equivalent to actual measurements ($\leq$ 1.3 Mbps lightweight real-time)

Actual measurements showed 4.7 MB/hour overall, approximately 2.35 MB per user for 30 minutes.

6.1.1 Fully Managed Cloud Pricing

Table 2 shows the approximate pricing for fully managed cloud in the Tokyo region.

Table 2: Data Transfer Pricing (Tokyo Region Approximate) Example: CloudFront

∼1TB	∼10TB	∼50TB	∼150TB
Free	$0.114/GB	$0.089/GB	$0.086/GB
∼500TB	∼1PB	∼5PB	5PB∼
$0.084/GB	$0.080/GB	$0.070/GB	$0.060/GB

Table 3: Other Charges

Item	Unit Price
HTTPS Requests	$0.010 / 10,000 req
Connection Hold Time	No charge

6.1.2 Monthly Cost Estimation by DAU

Table 4 shows the monthly cost estimation by DAU.

Table 4: Monthly Cost Estimation by DAU (1 Month)

DAU	Transfer Volume	Transfer Cost	Request Cost	Monthly Total
1K	70.5 GB	$0	$0.03	$0
10K	705 GB	$0	$0.30	$0
100K	7.05 TB	$690	$3.0	$693
1M	70.5 TB	$6,500	$30	$6,530

6.1.3 Advantages of the Proposed Method

The proposed method has the following operational advantages:

- QUIC congestion control window grows between Edge and UA
- Origin can focus solely on "object generation"
- Many public CDNs have minimal per-connection-time charges, though actual billing structures depend on the CDN (CloudFront was used in this experiment)
- Request charges are negligible compared to transfer charges
- SFU is not required, and scale-out via resolver.json on cloud storage is easy
- Network load on Origin is reduced, but CPU/IO load shifts to the Edge side
- Delivery of identical objects in live streaming has high affinity with CDN

6.1.4 Cost Comparison with Managed SFU Services

For reference, we estimate costs when using managed SFU services. Typical service pricing is as follows:

- Service (A): $0.004/min/participant
- Service (B): $0.001/min/participant

For 1M DAU with 1 session of 30 min/day, the total monthly minutes are 1,000,000 × 30 min × 30 days = 900,000,000 minutes.

Table 5 shows the cost comparison between the proposed method and managed SFU services.

Table 5: Cost Comparison: Proposed Method vs Managed SFU Services

DAU	Proposed Method	Service (A)	Service (B)
1K	$0	$3,600	$900
10K	$0	$36,000	$9,000
100K	$693	$360,000	$90,000
1M	$6,530	$3,600,000	$900,000

The proposed method achieves a cost difference of approximately 2 to 3 orders of magnitude compared to managed SFU services. This is due to the fact that the proposed method constructs HTTP/3-based real-time CDN in a fully managed configuration and can be operated with only transfer volume charges.

7. Comparative Experiment

This chapter presents quantitative comparisons between the proposed method and conventional session-oriented methods (WebRTC/SFU).

7.1 Experimental Environment

Comparative experiments were conducted in the following environment.

Table 6: Experimental Environment

Item	Specification
Edge Relay Proxy (Origin)	AWS EC2 m6i.large (Tokyo Region)
CDN	Amazon CloudFront (HTTP/3 enabled)
Client	iOS device, Native application (game)
Network	Mobile (4G LTE)
Media	OPUS 48kHz, Stereo
Comparison SFU	Custom SFU in production operation

7.2 Time To First Audio Byte (TTFAB) Measurement

Time To First Audio Byte (TTFAB) was measured. TTFAB is the time from when the UA initiates a connection until the first audio payload is received. For each method, the entire process from connection initiation to first audio data reception is the measurement target.

7.2.1 WebRTC New Connection Process

The process from new connection to first audio data reception in WebRTC is as follows:

(1) **SDP Offer generation**: Generate Offer SDP containing local media information
(2) **Send Offer via signaling**: Send Offer to peer via signaling server such as WebSocket
(3) **SDP Answer generation**: Peer receives Offer and generates Answer SDP
(4) **Return Answer via signaling**: Return Answer via signaling server
(5) **setLocalDescription / setRemoteDescription**: Set SDP on both sides
(6) **ICE candidate gathering**:
   - Host candidates: Local IP address (immediate)
   - Server Reflexive candidates: STUN (RTT dependent)
   - Relay candidates: TURN QUERY (RTT dependent)
   - Timeout waiting for each candidate type gathering (typically 3 to 10 seconds)
(7) **ICE connectivity check**: Execute STUN Binding Request/Response for each candidate pair, verifying connectable paths (timeout is in seconds for each candidate pair)
(8) **DTLS-SRTP handshake**: Establish encrypted session after ICE connection establishment (1 to 2 RTT)
(9) **MediaTrack establishment**: Start RTP/RTCP streams
(10) **First audio data reception** (measurement end)

In NAT environments, the time required for ICE candidate gathering and connectivity checking is dominant. Especially in Symmetric NAT environments, fallback via TURN relay occurs, requiring additional time.

7.2.2 Proposed Method Connection Process

The process from connection to first audio data reception in the proposed method is as follows:

(1) **QUIC handshake**: UA connects to CDN edge via HTTP/3 over QUIC (0-RTT or 1-RTT)
(2) **HTTP/3 request transmission**: Request containing `X-Publish-ID`, `X-Proxy-Offset` headers
(3) **CDN → Edge Relay Proxy**: CDN edge forwards to Edge Relay Proxy as HTTP/1.1 TCP Upstream
(4) **Octet stream return**: Edge Relay Proxy identifies stream by `X-Publish-ID` and returns stream as `application/octet-stream` from offset specified by `X-Proxy-Offset`
(5) **Delivery to UA via CDN edge**: First audio data reception (measurement end)

CDN edge handles caching and delivery, avoiding load concentration on Edge Relay Proxy. SDP exchange, ICE candidate gathering, and DTLS handshake are completely unnecessary, and audio streaming can be started with simple HTTP request/response only.

7.2.3 Measurement Results

Table 7 shows the TTFAB measurement results for each method. WebRTC measurements were conducted in NAT environment (via typical home router).

In WebRTC P2P, device-side SDP generation takes about 200ms, server-side SDP processing (including ICE/DTLS) takes about 200ms, and another 100 to 200ms until audio track OnOpen, resulting in TTFAB of 500 to 600ms even in the best case. When

Table 7: Comparison of Time To First Audio Byte (TTFAB)

Method	TTFAB	Notes
WebRTC P2P (best case)	500 to 600 ms	Device-SDP 200ms + SFU-SDP(incl. ICE/DTLS) 200ms + Track OnOpen 100 to 200ms
WebRTC P2P (NAT environment)	500 to 600 ms + several seconds	When candidate fallback occurs
Proposed method (HTTP/3 0-RTT)	180 ms	Reconnection to known edge
Proposed method (HTTP/3 1-RTT)	400 to 420 ms	New connection

candidate fallback occurs in NAT environment, this increases to seconds. In contrast, the proposed method received first audio data in 180ms with HTTP/3 0-RTT and 400 to 420ms with 1-RTT.

7.2.4 Discussion

The proposed method (0-RTT, 180ms) achieved approximately 3x faster TTFAB compared to WebRTC P2P (best case, 500 to 600ms). Furthermore, when candidate fallback occurs in NAT environment, WebRTC delays become on the order of seconds, so the difference expands by orders of magnitude. This difference is due to the following architectural differences:

- **P2P vs CDN**: WebRTC requires NAT traversal for P2P connection, while the proposed method connects directly to CDN edge with global IP, so NAT traversal issues do not occur
- **No negotiation required**: WebRTC requires capability negotiation via SDP exchange, while the proposed method has media format predetermined, so no negotiation is required
- **Stateless design**: The proposed method conveys state via HTTP headers with `X-Publish-ID` and `X-Proxy-Offset`, so server-side session state restoration is not required

7.3 Observed Characteristics

Based on the experimental results, the observed characteristics of the proposed method are summarized below.

Observation 1 (State Management Separation): In TTFAB measurement, the proposed method (0-RTT, 180ms) achieved approximately 3x faster first audio byte arrival compared to WebRTC P2P (best case, 500 to 600ms). While WebRTC requires complex processes including SDP exchange, ICE candidate gathering (including timeout waiting), DTLS handshake, and MediaTrack establishment, the proposed method can start streaming with simple HTTP requests using only `X-Publish-ID` and `X-Proxy-Offset` headers.

Observation 2 (CDN Utilization): In operational cost analysis, cost reduction of 2 to 3 orders of magnitude compared to managed SFU services was confirmed as shown in Table 5.

8. Use Case: Downstream

8.1 Audio Media Stream

Figure 5 shows the configuration. Audio encoded with OPUS is stored in PES/MPEGTS container and delivered as HTTP/3 octet stream. Transfer path is: Media Source Player → RTP → Edge Relay Proxy → HTTP/1 → CDN → HTTP/3 → UA.

8.2 Stream Control Plane

Figure 6 shows the configuration. JSON format event notifications are delivered through the same PES/MPEGTS path as media. Time synchronization between media and events becomes easy, and separate connections such as WebSocket are not required.

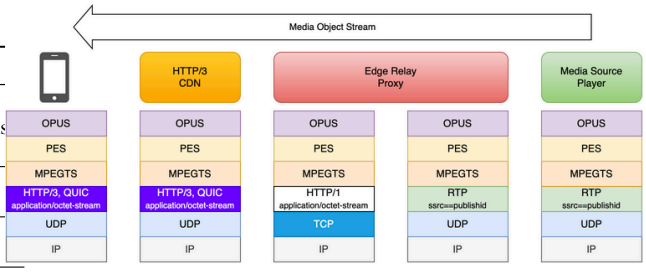

Fig. 5: Audio Media Object Stream

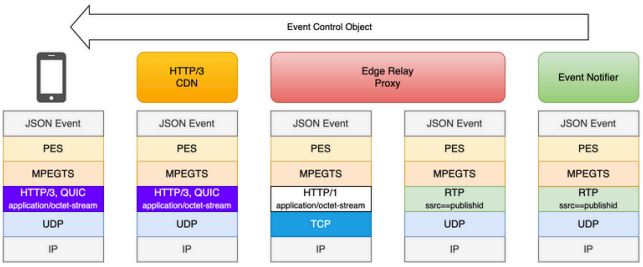

Fig. 6: Control Plane

9. Use Case: Upstream

9.1 Automatic Speech Recognition (ASR)

Figure 7 shows two configurations: UA-side and edge-side. UA-side (left) executes speech recognition on UA and sends textualized results as JSON. Edge-side (right) has UA send OPUS-encoded audio, and Edge Relay Proxy executes speech recognition.

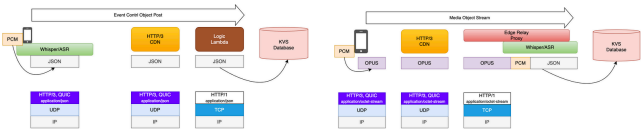

Fig. 7: Comparison of ASR Configurations (Left: UA-Side ASR, Right: Edge-Side ASR)

9.2 Control Plane

Figure 8 shows the configuration. JSON events are sent from UA through HTTP/3 CDN. Symmetric design to downstream enables bidirectional event-driven communication.

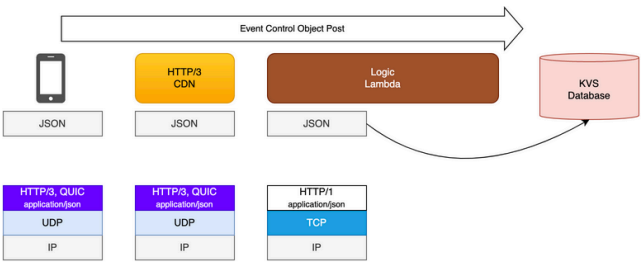

Fig. 8: Control Plane

10. Conclusion

10.1 Contributions of This Research

This research proposed an approach that redefines real-time media delivery not as session-oriented communication, but as a delivery problem of continuously updated objects. The main contributions of this research are as follows:

(1) **Redefinition of communication model**: Presented a new model that separates media state from transport sessions and manages it as object offsets.

(2) **Fast stream start**: Achieved approximately 3x faster TTFAB (0-RTT, 180ms) compared to WebRTC P2P (best case, 500 to 600ms). The difference becomes orders of magnitude when candidate fallback occurs in NAT environments. While WebRTC requires SDP exchange, ICE candidate gathering and connectivity checking (timeout waiting in seconds), DTLS handshake, and MediaTrack establishment, the proposed method can immediately start streaming from any offset with simple HTTP requests using only `X-Publish-ID` and `X-Proxy-Offset`.

(3) **Cost efficiency improvement**: Achieved cost reduction of 2 to 3 orders of magnitude compared to managed SFU services by utilizing CDN transfer volume billing model.

(4) **Design space expansion**: Overturned the conventional wisdom that "CDNs are unsuitable for real-time delivery" and presented a practical architecture for real-time media delivery using HTTP/3 CDN.

10.2 Scope of Application and Constraints

The proposed method is particularly suited for the following use cases:

- One-to-many large-scale live delivery
- Stable media viewing in mobile environments
- Commercial services emphasizing cost efficiency

On the other hand, there are the following constraints:

- For applications requiring ultra-low latency (below 10ms), WebRTC P2P can be superior when direct P2P connection is established within the same NAT or between hosts with global IPs. However, in typical network environments, Symmetric NAT and firewall restrictions often prevent direct P2P connectivity, resulting in TURN relay fallback. In such cases, the low-latency advantage of P2P is lost, limiting the scenarios where WebRTC's ultra-low latency can be achieved.
- Affected by session cache design at CDN edge

This research demonstrates new possibilities for real-time media delivery through object-centric design using HTTP/3 CDN.

11. Conflicts of Interest

The authors declare no conflicts of interest.

12. Data Availability Statement

The experimental data and measurement results presented in this study are available from the corresponding author upon reasonable request.

13. Author Contributions

Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data Curation, Writing-Original Draft, Writing-Review and Editing, Visualization, Supervision, Project administration: D.Sugisawa.

14. AI Disclosure

The authors used a large language model ChatGPT, OpenAI to assist in proofreading, grammar checking, and improving the clarity of expressions. The authors reviewed and are responsible for the final content.

15. Funding

This research received no external funding.

References

[1] C. Jennings, H. Boström, J.-I. Bruaroey: WebRTC 1.0: Real-Time Communication Between Browsers, W3C Recommendation (2021).
[2] R. Pantos, W. May: HTTP Live Streaming, RFC 8216, IETF (2017).
[3] ISO/IEC 23009-1:2022: Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats.
[4] ISO/IEC 23000-19:2020: Information technology — Multimedia application format (MPEG-A) — Part 19: Common media application format (CMAF) for segmented media.
[5] Apple Inc.: Enabling Low-Latency HTTP Live Streaming (HLS), Apple Developer Documentation (2019).
[6] M. Bishop: HTTP/3, RFC 9114, IETF (2022).
[7] J. Iyengar, M. Thomson: QUIC: A UDP-Based Multiplexed and Secure Transport, RFC 9000, IETF (2021).
[8] L. Curley, K. Pugin, S. Nandakumar, V. Vasiliev: Media over QUIC Transport, draft-ietf-moq-transport (work in progress), IETF (2024). https://datatracker.ietf.org/doc/draft-ietf-moq-transport
[9] K. Pugin, L. Curley: WARP – Segmented Live Video Transport, draft-lcurley-warp-02, IETF (2022). https://www.ietf.org/archive/id/draft-lcurley-warp-02.html
[10] K. Pugin, N. Garg, A. Frindell, J. Cenzano, J. Weissman: RUSH – Reliable (Unreliable) Streaming Protocol, draft-kpugin-rush-03, IETF (2025). https://www.ietf.org/archive/id/draft-kpugin-rush-03.html