Workgroup: Independent Submission
Internet-Draft: draft-lcurley-warp-02
Published: 24 October 2022
Intended Status: Informational
Expires: 27 April 2023
Authors: L. Curley K. Pugin S. Nandakumar
Twitch Meta Cisco

Warp - Segmented Live Media Transport

Abstract

This document defines the core behavior for Warp, a segmented live media transport protocol over QUIC. Media is split into segments based on the underlying media encoding and transmitted independently over QUIC streams. QUIC streams are prioritized based on the delivery order, allowing less important segments to be starved or dropped during congestion.

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 27 April 2023.

Table of Contents

1. Introduction

Warp is a live media transport protocol that utilizes the QUIC network protocol [QUIC].

*Section 2 covers the background and rationale behind Warp.

*Section 3 covers how media is encoded and split into segments.

*Section 4 covers how QUIC is used to transfer media.

*Section 5 covers how messages are encoded on the wire.

1.1. Terms 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.

Commonly used terms in this document are described below.

Bitstream: A continunous series of bytes.

Codec: A compression algorithm for audio or video.

Congestion: Packet loss and queuing caused by degraded or overloaded networks.

Consumer: A QUIC endpoint receiving media over the network. This could be the media player or middleware.

Container: A file format containing timestamps and the codec bitstream

Decoder: A endpoint responsible for a deflating a compressed media stream into raw frames.

Decode Timestamp (DTS): A timestamp indicating the order that frames/samples should be fed to the decoder.

Encoder: A component responsible for creating a compressed media stream out of raw frames.

Frame: An video image or group of audio samples to be rendered at a specific point in time.

I-frame:
A frame that does not depend on the contents of other frames; effectively an image.

Group of pictures (GoP): A I-frame followed by a sequential series of dependent frames.

Group of samples: A sequential series of audio samples starting at a given timestamp.

Player: A component responsible for presenting frames to a viewer based on the presentation timestamp.

Presentation Timestamp (PTS): A timestamp indicating when a frames/samples should be presented to the viewer.

Producer: A QUIC endpoint sending media over the network. This could be the media encoder or middleware.

Rendition: One or more tracks with the same content but different encodings.

Slice: A section of a video frame. There may be multiple slices per frame.

Track: An encoded bitstream, representing a single video/audio component that makes up the larger broadcast.

2. Motivation

2.1. Latency

In a perfect world, we could deliver live media at the same rate it is produced. The end-to-end latency of a broadcast would be fixed and only subject to encoding and transmission delays. Unfortunately, networks have variable throughput, primarily due to congestion.

Attempting to deliver media encoded at a higher bitrate than the network can support causes queuing. This queuing can occur anywhere in the path between the encoder and decoder. For example: the application, the OS socket, a wifi router, within an ISP, or generally anywhere in transit.

If nothing is done, new frames will be appended to the end of a growing queue and will take longer to arrive than their predecessors, increasing latency. Our job is to minimize the growth of this queue, and if necessary, bypass the queue entirely by dropping content.

The speed at which a media protocol can detect and respond to queuing determines the latency. We can generally classify existing media protocols into two categories based on the underlying network protocol:

- TCP-based media protocols (ex. RTMP, HLS, DASH) are popular due to their simplicity. Media is served/consumed in decode order while any networking is handled by the TCP layer. However, these protocols primarily see usage at higher latency targets due to their relatively slow detection and response to queuing.

- UDP-based media protocols (ex. RTP, WebRTC, SRT) can side-step the issues with TCP and provide lower latency with better queue management. However the media protocol is now responsible for fragmentation, congestion control, retransmissions, receiver feedback, reassembly, and more. This added complexity significantly raises the implementation difficulty and hurts interoperability.

A goal of this draft is to get the best of both worlds: a simple protocol that can still rapidly detect and respond to congestion. This is possible emergence of QUIC, designed to fix the shortcomings of TCP.

2.2. Universal

The media protocol ecosystem is fragmented; each protocol has it's own niche. Specialization is often a good thing, but we believe there's enough overlap to warrant consolidation.

For example, a service might simultaneously ingest via WebRTC, SRT, RTMP, and/or a custom UDP protocol depending on the broadcaster. The same service might then simultaneously distribute via WebRTC, LL-HLS, HLS, (or the DASH variants) and/or a custom UDP protocol depending on the viewer.

These media protocols are often radically different and not interoperable; requiring transcoding or transmuxing. This cost is further increased by the need to maintain separate stacks with different expertise requirements.

A goal of this draft is to cover a large spectrum of use-cases. Specifically:

- Consolidated contribution and distribution. The primary difference between the two is the ability to fanout. How does a CDN know how to forward media to N consumers and how does it reduce the encoded bitrate during congestion? A single protocol can cover both use-cases provided relays are informed on how to forward and drop media.

* A configurable latency versus quality trade-off. The producer (broadcaster) chooses how to encode and transmit media based on the desired user experience. Each consumer (viewer) chooses how long to wait for media based on their desired user experience and network. We want an experience that can vary from real-time and lossy for one viewer, to delayed and loss-less for another viewer, without separate encodings or protocols.

A related goal is to not reinvent how media is encoded. The same codec bitstream and container should be usable between different protocols.

2.3. Relays

The prevailing belief is that UDP-based protocols are more expensive and don't "scale". While it's true that UDP is more difficult to optimize than TCP, QUIC itself is proof that it is possible to reach performance parity. In fact even some TCP-based protocols (ex. RTMP) don't "scale" either and are exclusively used for contribution as a result.

The ability to scale a media protocol actually depends on relay support: proxies, caches, CDNs, SFUs, etc. The success of HTTP-based media protocols is due to the ability to leverage traditional HTTP CDNs.

It's difficult to build a CDN for media protocols that were not designed with relays in mind. For example, an relay has to parse the underlying codec to determine which RTP packets should be dropped first, and the decision is not deterministic or consistent for each hop. This is the fatal flaw of many UDP-based protocols.

A goal of this draft is to treat relays as first class citizens. Any identification, reliability, ordering, prioritization, caching, etc is written to the wire in a header that is easy to parse. This ensures that relays can easily route/fanout media to the final destination. This also ensures that congestion response is consistent at every hop based on the preferences of the media producer.

3. Segments

Warp works by splitting media into segments that can be transferred over QUIC streams.

* The encoder determines how to fragment the encoded bitstream into segments (Section 3.1).

* Segments are assigned an intended delivery order that should be obeyed during congestion (Section 3.2)

* Segments can be dependent on other segments, in which case reordering is required (Section 3.3).

* The decoder receives each segment and skips any segments that do not arrive in time (Section 3.4).

3.1. Media

An encoder produces one or more codec bitstreams for each track. The decoder processes the codec bitstreams in the same order they were produced, with some possible exceptions based on the encoding. See the appendix for an overview of media encoding (Section 8).

Warp works by fragmenting the bitstream into segments that can be transmitted somewhat independently. Depending on how the segments are fragmented, the decoder has the ability to safely drop media during congestion. See the appendix for fragmentation examples (Section 9)

A segment:

* MUST contain a single track.

* MUST be in decode order. This means an increasing DTS.

* MAY contain any number of frames/samples.

* MAY have gaps between frames/samples.

* MAY overlap with other segments. This means timestamps may be interleaved between segments.

* MAY reference frames in other segments, but only if listed as a dependency.

Segments are encoded using fragmented MP4 [ISOBMFF]. This is necessary to store timestamps and various metadata depending on the codec. A future draft of Warp may specify other container formats.

3.2. Delivery Order

Media is produced with an intended order, both in terms of when media should be presented (PTS) and when media should be decoded (DTS). As stated in motivation (Section 2.1), the network is unable to maintain this ordering during congestion without increasing latency.

The encoder determines how to behave during congestion by assigning each segment a numeric delivery order. The delivery order SHOULD be followed when possible to ensure that the most important media is

delivered when throughput is limited. Note that the contents within each segment are still delivered in order; this delivery order only applies to the ordering between segments.

A segment MUST NOT have a smaller delivery order than a segment it depends on. Delivering segments out of dependency order will increase latency and can cause artifacting when memory limits are tight. This is especially problematic and can cause a deadlock if the receiver does not release flow control until dependencies are received.

A sender MUST send each segment over a dedicated QUIC stream. The QUIC library should support prioritization (Section 4.3) such that streams are transmitted in delivery order.

A receiver MUST NOT assume that segments will be received in delivery order for a number of reasons:

* Newly encoded segments MAY have a smaller delivery order than outstanding segments.

* Packet loss or flow control MAY delay the delivery of individual streams.

* The sender might not support QUIC stream prioritization.

3.3. Dependencies

Media encoding uses references to improve the compression. This creates hard and soft dependencies that need to be respected by the transport. See the appendex for an overview of media encoding (Section 8).

A segment MAY depend on any number of other segments. The encoder MUST indicate these dependecies on the wire via the HEADERS message (Section 5.1).

The sender SHOULD NOT use this list of dependencies to determine which segment to transmit next. The sender SHOULD use the delivery order instead, which MUST respect dependencies.

The decoder SHOULD process segments according to their dependencies. This means buffering a segment until the relevent timestamps have been processed in all dependencies. A decoder MAY drop dependencies at the risk of producing decoding errors and artifacts.

3.4. Decoder

The decoder will receive multiple segments in parallel and out of order.

Segments arrive in delivery order, but media usually needs to be processed in decode order. The decoder SHOULD use a buffer to reassmble segments into decode order and it SHOULD skip segments after a configurable duration. The amount of time the decoder is willing to wait for a segment (buffer duration) is what ultimately determines the end-to-end latency.

Segments MUST synchronize frames within and between tracks using presentation timestamps within the container. Segments are NOT REQUIRED to be aligned and the decoder MUST be prepared to skip over any gaps.

4. QUIC

4.1. Establishment

A connection is established using WebTransport [WebTransport].

To summarize: The client issues a HTTP CONNECT request with the intention of establishing a new WebTransport session. The server returns an 200 OK response if the WebTransport session has been established, or an error status otherwise.

A WebTransport session exposes the basic QUIC service abstractions. Specifically, either endpoint may create independent streams which are reliably delivered in order until canceled.

WebTransport can currently operate via HTTP/3 and HTTP/2, using QUIC or TCP under the hood respectively. As mentioned in the motivation (Section 2) section, TCP introduces head-of-line blocking and will result in a worse experience. It is RECOMMENDED to use WebTransport over HTTP/3.

The application SHOULD use the CONNECT request for authentication. For example, including a authentication token and some identifier in the path.

4.2. Streams

Warp endpoints communicate over unidirectional QUIC streams. The application MAY use bidirectional QUIC streams for other purposes.

A stream consists of sequential messages. See messages (Section 5) for the list of messages and their encoding. These are similar to QUIC and HTTP/3 frames, but called messages to avoid the media terminology.

Each stream MUST start with a HEADERS message (Section 5.1) to indicates how the stream should be transmitted.

Messages SHOULD be sent over the same stream if ordering is desired. For example, PAUSE and PLAY messages SHOULD be sent on the same stream to avoid a race.

4.3. Prioritization

Warp utilizes stream prioritization to deliver the most important content during congestion.

The encoder may assign a numeric delivery order to each stream (Section 3.2) This is a strict prioritization scheme, such that any available bandwidth is allocated to streams in ascending priority order. The sender SHOULD prioritize streams based on the delivery order. If two streams have the same delivery order, they SHOULD receive equal bandwidth (round-robin).

QUIC supports stream prioritization but does not standardize any mechanisms; see Section 2.3 in [QUIC]. In order to support prioritization, a QUIC library MUST expose a API to set the priority of each stream. This is relatively easy to implement; the next QUIC packet should contain a STREAM frame for the next pending stream in priority order.

The sender MUST respect flow control even if means delivering streams out of delivery order. It is OPTIONAL to prioritize retransmissions.

4.4. Cancellation

A QUIC stream MAY be canceled at any point with an error code. The producer does this via a RESET_STREAM frame while the consumer requests cancellation with a STOP_SENDING frame.

When using order, lower priority streams will be starved during congestion, perhaps indefinitely. These streams will consume resources and flow control until they are canceled. When nearing resource limits, an endpoint SHOULD cancel the lowest priority stream with error code 0.

The sender MAY cancel streams in response to congestion. This can be useful when the sender does not support stream prioritization.

4.5. Relays

Warp encodes the delivery information for each stream via a HEADERS frame (Section 5.1). This MUST be at the start of each stream so it is easy for a relay to parse.

A relay SHOULD prioritize streams (Section 4.3) based on the delivery order. A relay MAY change the delivery order, in which case it SHOULD update the value on the wire for future hops.

A relay that reads from a stream and writes to stream in order will introduce head-of-line blocking. Packet loss will cause stream data to be buffered in the QUIC library, awaiting in order delivery, which will increase latency over additional hops. To mitigate this, a relay SHOULD read and write QUIC stream data out of order subject to flow control limits. See section 2.2 in [QUIC].

4.6. Congestion Control

As covered in the motivation section (Section 2), the ability to prioritize or cancel streams is a form of congestion response. It's equally important to detect congestion via congestion control, which is handled in the QUIC layer [QUIC-RECOVERY].

Bufferbloat is caused by routers queueing packets for an indefinite amount of time rather than drop them. This latency significantly reduces the ability for the application to prioritize or drop media in response to congestion. Senders SHOULD use a congestion control algorithm that reduces this bufferbloat (ex. [BBR]). It is NOT RECOMMENDED to use a loss-based algorithm (ex. [NewReno]) unless the network fully supports ECN.

Live media is application-limited, which means that the encoder determines the max bitrate rather than the network. Most TCP congestion control algorithms will only increase the congestion window if it is full, limiting the upwards mobility when application-limited. Senders SHOULD use a congestion control algorithm that is designed for application-limited flows (ex. GCC). Senders MAY periodically pad the connection with QUIC PING frames to fill the congestion window.

4.7. Termination

The QUIC connection can be terminated at any point with an error code.

The media producer MAY terminate the QUIC connection with an error code of 0 to indicate the clean termination of the broadcast. The application SHOULD use a non-zero error code to indicate a fatal error.

Code	Reason
0x0	Broadcast Terminated
0x1	GOAWAY (Section 5.4)

Table 1

5. Messages

Messages consist of a type identifier followed by contents, depending on the message type.

TODO document the encoding

ID	Messages
0x0	HEADERS (Section 5.1)
0x1	SEGMENT (Section 5.2)
0x2	APP (Section 5.3)
0x10	GOAWAY (Section 5.4)

Table 2

5.1. HEADERS

The HEADERS message contains information required to deliver, cache, and forward a stream. This message SHOULD be parsed and obeyed by any Warp relays.

*id. An unique identifier for the stream. This field is optional and MUST be unique if set.

*order. An integer indicating the delivery order (Section 3.2). This field is optional and the default value is 0.

*depends. An list of dependencies by stream identifier (Section 3.3). This field is optional and the default value is an empty array.

5.2. SEGMENT

A SEGMENT message consists of a segment in a fragmented MP4 container.

Each segment MUST start with an initialization fragment, or MUST depend on a segment with an initialization fragment. An initialization fragment consists of a File Type Box (ftyp) followed by a Movie Box (moov). This Movie Box (moov) consists of Movie Header Boxes (mvhd), Track Header Boxes (tkhd), Track Boxes (trak), followed by a final Movie Extends Box (mvex). These boxes MUST NOT contain any samples and MUST have a duration of zero. Note that a Common Media Application Format Header [CMAF] meets all these requirements.

Each segment MAY have a Segment Type Box (styp) followed by any number of media fragments. Each media fragment consists of a Movie Fragment Box (moof) followed by a Media Data Box (mdat). The Media Fragment Box (moof) MUST contain a Movie Fragment Header Box (mfhd) and Track Box (trak) with a Track ID (track_ID) matching a Track Box

in the initialization fragment. Note that a Common Media Application Format Segment [CMAF] meets all these requirements.

Media fragments can be packaged at any frequency, causing a trade-off between overhead and latency. It is RECOMMENDED that a media fragment consists of a single frame to minimize latency.

5.3. APP

The APP message contains arbitrary contents. This is useful for metadata that would otherwise have to be shoved into the media bitstream.

Relays MUST NOT differentiate between streams containing SEGMENT and APP frames. The same forwarding and caching behavior applies to both as specified in theHEADERS frame.

5.4. GOAWAY

The GOAWAY message is sent by the server to force the client to reconnect. This is useful for server maintenance or reassignments without severing the QUIC connection. The server MAY be a producer or consumer.

The server:

- MAY initiate a graceful shutdown by sending a GOAWAY message.
- MUST close the QUIC connection after a timeout with the GOAWAY error code (Section 4.7).
- MAY close the QUIC connection with a different error code if there is a fatal error before shutdown.
- SHOULD wait until the GOAWAY message and any pending streams have been fully acknowledged, plus an extra delay to ensure they have been processed.

The client:

- MUST establish a new WebTransport session to the provided URL upon receipt of a GOAWAY message.
- SHOULD establish the connection in parallel which MUST use different QUIC connection.
- SHOULD remain connected for two servers for a short period, processing segments from both in parallel.

6. Security Considerations

6.1. Resource Exhaustion

Live media requires significant bandwidth and resources. Failure to set limits will quickly cause resource exhaustion.

Warp uses QUIC flow control to impose resource limits at the network layer. Endpoints SHOULD set flow control limits based on the anticipated media bitrate.

The media producer prioritizes and transmits streams out of order. Streams might be starved indefinitely during congestion. The producer and consumer MUST cancel a stream, preferably the lowest priority, after reaching a resource limit.

7. IANA Considerations

TODO

8. Appendix A. Video Encoding

In order to transport media, we first need to know how media is encoded. This section is an overview of media encoding.

8.1. Tracks

A broadcast consists of one or more tracks. Each track has a type (audio, video, caption, etc) and uses a corresponding codec. There may be multiple tracks, including of the same type for a number of reasons.

For example:

*A track for each codec.

*A track for each resolution and bitrate.

*A track for each language.

*A track for each camera feed.

Tracks can be muxed together into a single container or stream. The goal of Warp is to independently deliver tracks, and even parts of a track, so this is not allowed. Each Warp segment MUST contain a single track.

8.2. Init

Media codecs have a wide array of configuration options. For example, the resolution, the color space, the features enabled, etc.

Before playback can begin, the decoder needs to know the configuration. This is done via a short payload at the very start of the media file. The initialization payload MAY be cached and reused between segments with the same configuration.

8.3. Video

Video is a sequence of pictures (frames) with a presentation timestamp (PTS).

An I-frame is a frame with no dependencies and is effectively an image file. These frames are usually inserted at a frequent interval to support seeking or joining a live stream. However they can also improve compression when used at scene boundaries.

A P-frame is a frame that references on one or more earlier frames. These frames are delta-encoded, such that they only encode the changes (motion). This result in a massive file size reduction for most content outside of few notorious cases (ex. confetti).

A common encoding structure is to only reference the previous frame, as it is simple and minimizes latency:

```
I <- P <- P <- P   I <- P <- P <- P   I <- P ...
```

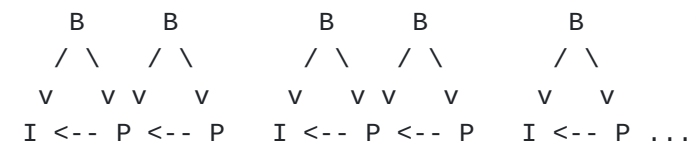
There is no such thing as an optimal encoding structure. Encoders tuned for the best quality will produce a tangled spaghetti of references. Encoders tuned for the lowest latency can avoid reference frames to allow more to be dropped.

8.3.1. B-Frames

The goal of video codecs is to maximize compression. One of the improvements is to allow a frame to reference later frames.

A B-frame is a frame that can reference one or more frames in the future, and any number of frames in the past. These frames are more difficult to encode/decode as they require buffering and reordering.

A common encoding structure is to use B-frames in a fixed pattern. Such a fixed pattern is not optimal, but it's simpler for hardware encoding:



8.3.2. Timestamps

Each frame is assigned a presentation timestamp (PTS), indicating when it should be shown relative to other frames.

The encoder outputs the bitstream in decode order, which means that each frame is output after its references. This makes it easier for the decoder as all references are earlier in the bitstream and can be decoded immediately.

However, this causes problems with B-frames because they depend on a future frame, and some reordering has to occur. In order to keep track of this, frames have a decode timestamp (DTS) in addition to a presentation timestamp (PTS). A B-frame will have higher DTS value that its dependencies, while PTS and DTS will be the same for other frame types.

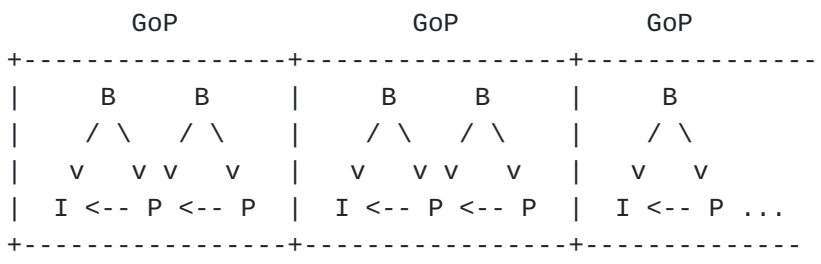
For the example above, this would look like:

```
     0 1 2 3 4 5 6 7 8 9 10
PTS: I B P B P I B P B P B
DTS: I   PB  PBI   PB  PB
```

B-frames add latency because of this reordering so they are usually not used for conversational latency.

8.3.3. Group of Pictures

A group of pictures (GoP) is an I-frame followed by any number of frames until the next I-frame. All frames MUST reference, either directly or indirectly, only the most recent I-frame.

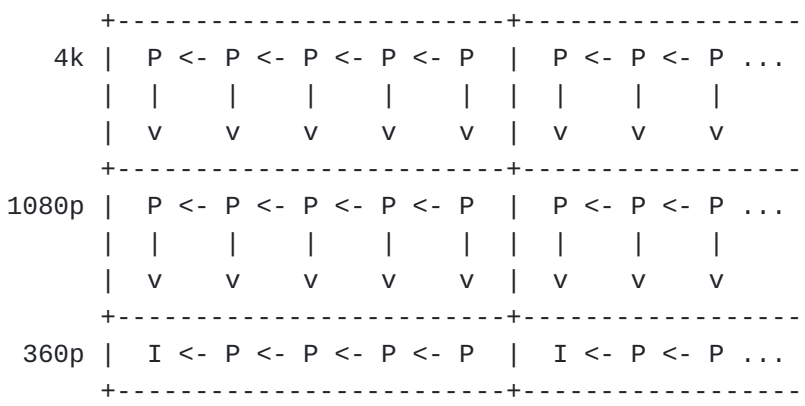


This is a useful abstraction because GoPs can always be decoded independently.

8.3.4. Scalable Video Coding

Some codecs support scalable video coding (SVC), in which the encoder produces multiple bitstreams in a hierarchy. This layered coding means that dropping the top layer degrades the user experience in a configured way. Examples include reducing the resolution, picture quality, and/or frame rate.

Here is an example SVC encoding with 3 resolutions:



8.4. Audio

Audio is dramatically simpler than video as it is not typically delta encoded. Audio samples are grouped together (group of samples) at a configured rate, also called a "frame".

The encoder spits out a continuous stream of samples (S):

S S S S S S S S S S S S ...

9. Appendix B. Segment Examples

Warp offers a large degree of flexibility on how segments are fragmented and prioritized. There is no best solution; it depends on the desired complexity and user experience.

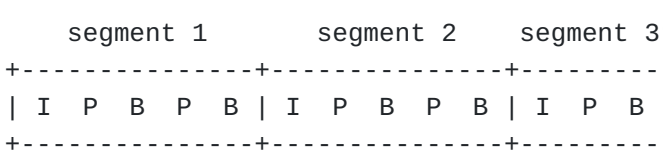
This section provides a summary of some options available.

9.1. Video

9.1.1. Group of Pictures

A group of pictures (GoP) is consists of an I-frame and all frames that directly or indirectly reference it (Section 8.3.3). The tail of a GoP can be dropped without causing decode errors, even if the encoding is otherwise unknown, making this the safest option.

It is RECOMMENDED that each segment consist of a single GoP. For example:

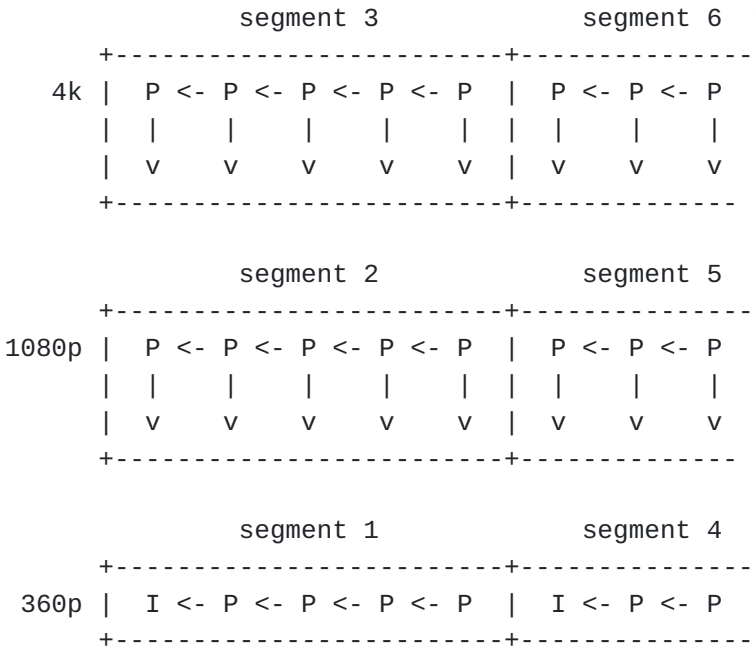


Depending on the video encoding, this approach may introduce unnecessary ordering and dependencies. A better option may be available below.

9.1.2. Scalable Video Coding

Some codecs support scalable video coding (SVC), in which the encoder produces multiple bitstreams in a hierarchy (Section 8.3.4).

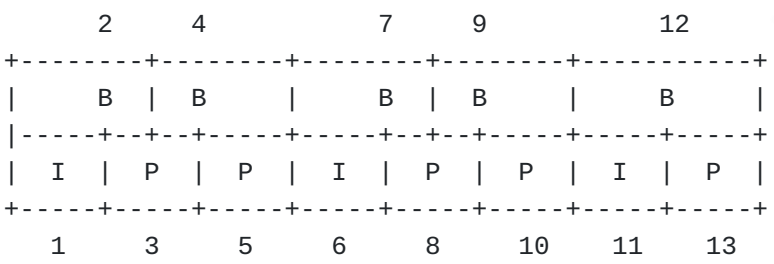
When SVC is used, it is RECOMMENDED that each segment consist of a single layer and GoP. For example:



9.1.3. Frames

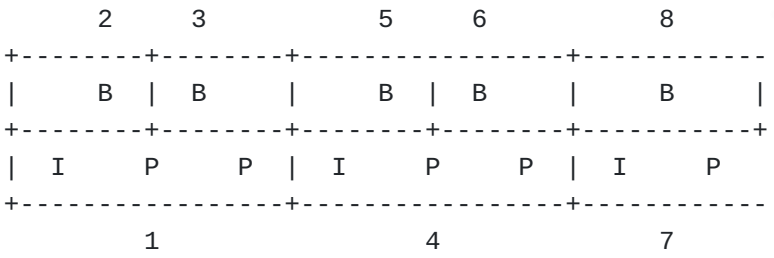
With full knowledge of the encoding, the encoder MAY can split a GoP into multiple segments based on the frame. However, this is highly dependent on the encoding, and the additional complexity might not improve the user experience.

For example, we could split our example B-frame structure (Section 8.3.1) into 13 segments:



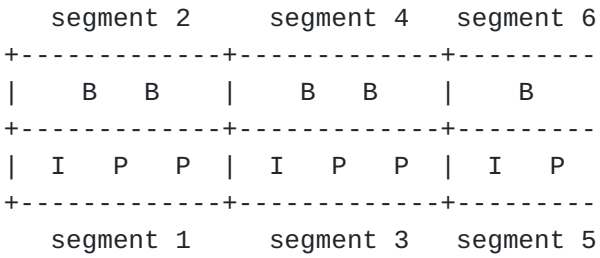
To reduce the number of segments, segments can be merged with their dependency. QUIC streams will deliver each segment in order so this produces the same result as reordering within the application.

The same GoP structure can be represented using eight segments:



We can further reduce the number of segments by combining frames that don't depend on each other. The only restriction is that frames can only reference frames earlier in the segment, or within a dependency segment. For example, non-reference frames can have their own segment so they can be prioritized or dropped separate from reference frames.

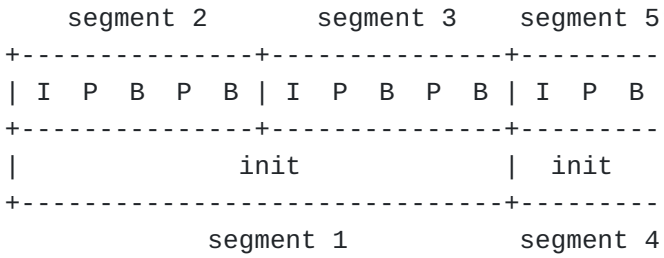
The same GoP structure can also be represented using six segments, although we've removed the ability to drop individual B-frames:



9.1.4. Init

Initialization data (Section 8.2) is required to initialize the decoder. Each segment MAY start with initialization data although this adds overhead.

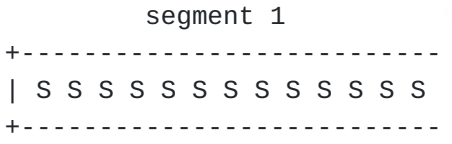
Instead, it is RECOMMENDED to create a init segment. Each media segment can then depend on the init segment to avoid the redundant overhead. For example:



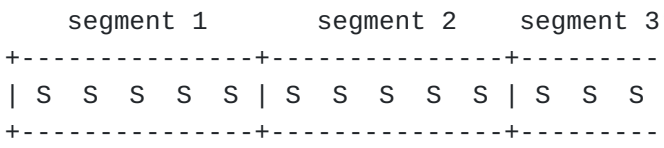
9.2. Audio

Audio (Section 8.4) is much simpler than video so there's fewer options.

The simplest configuration is to use a single segment for each audio track. This may seem inefficient given the ease of dropping audio samples. However, the audio bitrate is low and gaps cause quite a poor user experience, when compared to video.

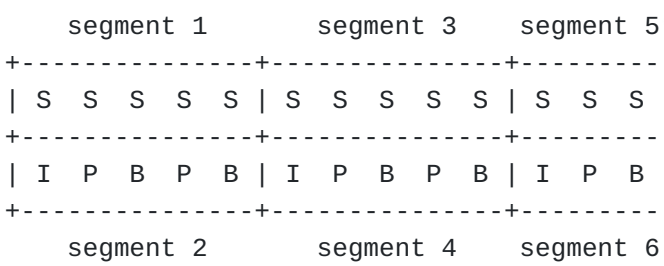


An improvement is to periodically split audio samples into separate segments. This gives the consumer the ability to skip ahead during severe congestion or temporary connectivity loss.



This frequency of audio segments is configurable, at the cost of additional overhead. It's NOT RECOMMENDED to create a segment for each audio frame because of this overhead.

Since video can only recover from severe congestion with an I-frame, so there's not much point recovering audio at a separate interval. It is RECOMMENDED to create a new audio segment at each video I-frame.



9.3. Delivery Order

The delivery order (Section 3.2 depends on the desired user experience during congestion:

- *if media should be skipped: delivery order = PTS
- *if media should not be skipped: delivery order = -PTS
- *if video should be skipped before audio: audio delivery order < video delivery order

The delivery order may be changed if the content changes. For example, switching from a live stream (skippable) to an advertisement (unskippable).

Contributors

- *Alan Frindell
- *Charles Krasic
- *Cullen Jennings
- *James Hurley
- *Jordi Cenzano
- *Mike English

References

Normative References

[ISOBMFF] "Information technology — Coding of audio-visual objects — Part 12: ISO Base Media File Format", December 2015.

[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/info/rfc9000>.

[QUIC-RECOVERY] Iyengar, J., Ed. and I. Swett, Ed., "QUIC Loss Detection and Congestion Control", RFC 9002, DOI 10.17487/RFC9002, May 2021, <https://www.rfc-editor.org/info/rfc9002>.

[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/info/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/info/rfc8174>.

[WebTransport] Frindell, A., Kinnear, E., and V. Vasiliev, "WebTransport over HTTP/3", Work in Progress, Internet-Draft, draft-ietf-webtrans-http3-03, 6 July 2022, <https://www.ietf.org/archive/id/draft-ietf-webtrans-http3-03.txt>.

Informative References

[BBR] Cardwell, N., Cheng, Y., Yeganeh, S. H., Swett, I., and V. Jacobson, "BBR Congestion Control", Work in Progress, Internet-Draft, draft-cardwell-iccrg-bbr-congestion-control-02, 7 March 2022, <https://www.ietf.org/archive/id/draft-cardwell-iccrg-bbr-congestion-control-02.txt>.

[CMAF] "Information technology -- Multimedia application format (MPEG-A) -- Part 19: Common media application format (CMAF) for segmented media", March 2020.

[NewReno] Henderson, T., Floyd, S., Gurtov, A., and Y. Nishida, "The NewReno Modification to TCP's Fast Recovery Algorithm", RFC 6582, DOI 10.17487/RFC6582, April 2012, <https://www.rfc-editor.org/info/rfc6582>.

Authors' Addresses

Luke Curley
Twitch

Email: kixelated@gmail.com

Kirill Pugin
Meta

Email: ikir@meta.com

Suhas Nandakumar
Cisco

Email: snandaku@cisco.com