Workgroup: Independent Submission
Internet-Draft: draft-lcurley-warp-04
Published: 13 March 2023
Intended Status: Informational
Expires: 14 September 2023
Authors: L. Curley K. Pugin S. Nandakumar V. Vasiliev
Twitch Meta Cisco Google

Warp - Live Media Transport over QUIC

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

*Section 3 covers the background and rationale behind Warp.

*Section 2.1 covers how media is fragmented into objects.

*Section 5 covers how QUIC is used to transfer media.

*Section 6 covers how messages are encoded on the wire.

*Section 8 covers how media tracks are packaged.

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.

Client: The party initiating a Warp session.

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.

Server: The party accepting an incoming Warp session.

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

Track: An encoded bitstream, representing a single media component (ex. audio, video, subtitles) that makes up the larger broadcast.

Variant: A track with the same content but different encoding as another track. For example, a different bitrate, codec, language, etc.

1.2. Notational Conventions

This document uses the conventions detailed in Section 1.3 of [RFC9000] when describing the binary encoding.

This document also defines an additional field type for binary data:

x (b): Indicates that x consists of a variable length integer, followed by that many bytes of binary data.

2. Model

2.1. Objects

The basic element of Warp is an *object*. An object is a single addressable cacheable unit whose payload is a sequence of bytes. An object **MAY** depend on other objects to be decoded. An object **MUST** belong to a group Section 2.2. Objects carry associated metadata such as priority, TTL or other information usable by a relay, but relays **MUST** treat object payloads as opaque.

DISCUSS: Can an object be partially decodable by an endpoint?

Authors agree that an object is always partially *forwardable* by a relay but disagree on whether a partial object can be used by a receiving endpoint.

Option 1: A receiver **MAY** start decoding an object before it has been completely received

Example: sending an entire GOP as a single object. A receiver can decode the GOP from the beginning without having the entire object present, and the object's tail could be dropped. Sending a GOP as a group of not-partially-decodable objects might incur additional overhead on the wire and/or additional processing of video segments at a sender to find object boundaries.

Partial decodability could be another property of an object.

Option 2: A receiver **MUST NOT** start decoding an object before it has completely arrived

Objects could be end-to-end encrypted and the receiver might not be able to decrypt or authenticate an object until it is fully present. Allowing Objects to span more than one useable unit may create more than one viable application mapping from media to wire format, which could be confusing for protocol users.

2.2. Groups

An object group is a sequence of media objects. Beginning of an object group can be used as a point at which the receiver can start consuming a track without having any other object groups available. Object groups have an ID that identifies them uniquely within a track.

DISCUSS: We need to determine what are the exact requirements we need to impose on how the media objects depend on each other. Such requirements would need to address the use case (a join point), while being flexible enough to accomodate scenarios like B-frames and temporal scaling.

2.3. Track

A media track in Warp is a combination of *an init object* and a sequence of media object groups. An init object is a format-specific self-contained description of the track that is required to decode any media object contained within the track, but can also be used as the metadata for track selection. If two media tracks carry semantically equivalent but differently encoded media, they are referred to as *variants* of each other.

2.4. Track Bundle

A track bundle is a collection of tracks intended to be delivered together. Objects within a track bundle may be prioritized relative to each other via the delivery order property. This allows objects to be prioritized within a track (ex. newer > older) and between tracks (ex. audio > video). The track bundle contains a catalog indicating the available tracks.

2.5. Session

A WebTransport session is established for each track bundle. The client issues a CONNECT request with a URL which the server uses for identification and authentication. All control messages and prioritization occur within the context of a single WebTransport session, which means a single track bundle. Multiple WebTransport sessions may be pooled over a single QUIC connection for efficiency.

2.6. Example

As an example, consider a scenario where example.org hosts a simple live stream that anyone can subscribe to. That live stream would be a single track bundle, accessible via the WebTransport URL: https://example.org/livestream. In a simple scenario, the track bundle would contain only two media tracks, one with audio and one with video. In a more complicated scenario, the track bundle could multiple tracks with different formats, encodings, bitrates, and quality levels, possibly for the same content. The receiver learns about each available track within the bundle via the catalog, and can choose to subscribe to a subset.

3. Motivation

3.1. 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 with the emergence of QUIC, designed to fix the shortcomings of TCP.

3.2. 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.

3.3. 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.

4. Objects

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

* The encoder determines how to fragment the encoded bitstream into objects (Section 4.1).
* Objects are assigned an intended delivery order that should be obeyed during congestion (Section 4.2)
* The decoder receives each objects and skips any objects that do not arrive in time (Section 4.3).

4.1. 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 11).

Warp works by fragmenting the bitstream into objects that can be transmitted somewhat independently. Depending on how the objects are fragmented, the decoder has the ability to safely drop media during congestion. See the appendix for fragmentation examples (Section 12)

A media object:

* **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 objects. This means timestamps may be interleaved between objects.

Media objects are encoded using a specified container (Section 8).

4.2. 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 3.1), the network is unable to maintain this ordering during congestion without increasing latency.

The encoder determines how to behave during congestion by assigning each object 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 object are still delivered in order; this delivery order only applies to the ordering between objects.

A sender **MUST** send each object over a dedicated QUIC stream. The QUIC library should support prioritization (Section 5.3) such that streams are transmitted in delivery order.

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

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

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

* The sender might not support QUIC stream prioritization.

4.3. Decoder

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

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

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

5. QUIC

5.1. Establishment

A connection is established using WebTransport [WebTransport].

To summarize: The client issues a HTTP CONNECT request to a URL. The server returns an "200 OK" response to establish the WebTransport session, or an error status code 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 3) section, TCP introduces head-of-line blocking and will result in a worse experience. It is **RECOMMENDED** to use WebTransport over HTTP/3.

5.1.1. CONNECT

The server uses the HTTP CONNECT request for identification and authorization of a track bundle. The specific mechanism is left up to the application. For example, an identifier and authentication token could be included in the path.

The server **MAY** return an error status code for any reason, for example a 403 when the client is forbidden. Otherwise the server **MUST** respond with a "200 OK" to establish the WebTransport session.

5.2. Streams

Warp endpoints communicate over QUIC streams. Every stream is a sequence of messages, framed as described in Section 6.

The first stream opened is a client-initiated bidirectional stream where the peers exchange SETUP messages (Section 6.1). The subsequent streams **MAY** be either unidirectional and bidirectional. For exchanging media, an application would typically send a unidirectional stream containing a single OBJECT message (Section 6.2).

Messages **SHOULD** be sent over the same stream if ordering is desired.

5.3. Prioritization

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

The producer may assign a numeric delivery order to each object (Section 4.2) 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.

5.4. 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.

5.5. Relays

Warp encodes the delivery information for a stream via OBJECT headers (Section 6.2).

A relay **SHOULD** prioritize streams (Section 5.3) 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].

5.6. Congestion Control

As covered in the motivation section (Section 3), 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.

5.7. Termination

The WebTransport session can be terminated at any point with CLOSE_WEBTRANSPORT_SESSION capsule, consisting of an integer code and string message.

The application **MAY** use any error message and **SHOULD** use a relevant code, as defined below:

Code	Reason
0x0	Session Terminated
0x1	Generic Error
0x2	Unauthorized
0x10	GOAWAY

Table 1

* Session Terminated No error occured, however the endpoint no longer desires to send or receive media.

* Generic Error An unclassified error occured.

*Unauthorized: The endpoint breached an agreement, which **MAY** have been pre-negotiated by the application.

*GOAWAY: The endpoint successfully drained the session after a GOAWAY was initiated (Section 6.5).

6. Messages

Both unidirectional and bidirectional Warp streams are sequences of length-deliminated messages.

```
Warp Message {
  Message Type (i),
  Message Length (i),
  Message Payload (..),
}
```

Figure 1: Warp Message

The Message Length field contains the length of the Message Payload field in bytes. A length of 0 indicates the message is unbounded and continues until the end of the stream.

ID	Messages
0x0	OBJECT (Section 6.2)
0x1	SETUP (Section 6.1)
0x2	CATALOG (Section 6.3)
0x3	SUBSCRIBE (Section 6.4)
0x10	GOAWAY (Section 6.5)

Table 2

6.1. SETUP

The SETUP message is the first message that is exchanged by the client and the server; it allows the peers to establish the mutually supported version and agree on the initial configuration. It is a sequence of key-value pairs called *SETUP parameters*; the semantics and the format of individual parameter values **MAY** depend on what party is sending it.

The wire format of the SETUP message is as follows:

```
SETUP Parameter {
  Parameter Key (i),
  Parameter Value Length (i),
  Parameter Value (..),
}

Client SETUP Message Payload {
  Number of Supported Versions (i),
  Supported Version (i) ...,
  SETUP Parameters (..) ...,
}

Server SETUP Message Payload {
  Selected Version (i),
  SETUP Parameters (..) ...,
}
```

Figure 2: Warp SETUP Message

The Parameter Value Length field indicates the length of the Parameter Value.

The client offers the list of the protocol versions it supports; the server **MUST** reply with one of the versions offered by the client. If the server does not support any of the versions offered by the client, or the client receives a server version that it did not offer, the corresponding peer **MUST** close the connection.

The SETUP parameters are described in the Section 7 section.

6.2. OBJECT

A OBJECT message contains a single media object associated with a specified track, as well as associated metadata required to deliver, cache, and forward it.

The format of the OBJECT message is as follows:

```
OBJECT Message {
  Track ID (i),
  Group Sequence (i),
  Object Sequence (i),
  Object Delivery Order (i),
  Object Payload (b),
}
```

Figure 3: Warp OBJECT Message

* Track ID: The track identifier as declared in CATALOG (Section 6.3).

* Group Sequence : An integer always starts at 0 and increases sequentially at the original media publisher. Group sequences are scoped under a Track.

* Object Sequence: An integer always starts at 0 with in a Group and increases sequentially. Object Sequences are scoped to a Group.

* Object Delivery Order: An integer indicating the object delivery order (Section 4.2).

* Object Payload: The format depends on the track container (Section 8). This is a media bitstream intended for the decoder and **SHOULD NOT** be processed by a relay.

6.3. CATALOG

The sender advertises tracks via the CATALOG message. The receiver can then SUBSCRIBE to the indiciated tracks by ID.

The format of the CATALOG message is as follows:

```
CATALOG Message {
  Track Count (i),
  Track Descriptors (..)
}
```

Figure 4: Warp CATALOG Message

* Track Count: The number of tracks within the catalog.

For each track, there is a track descriptor with the format:

```
Track Descriptor {
  Track ID (i),
  Container Format (i),
  Container Init Payload (b)
}
```

Figure 5: Warp Track Descriptor

* Track ID: A unique identifier for the track within the track bundle.

* Container Format: The container format as defined in Section 8.

* Container Init Payload: A container-specific payload as defined in Section 8. This contains base information required to decode OBJECT messages, such as codec parameters.

An endpoint **MUST NOT** send multiple CATALOG messages. A future draft will add the ability to add/remove/update tracks.

6.4. SUBSCRIBE

After receiving a CATALOG message (Section 6.3, the receiver sends a SUBSCRIBE message to indicate that it wishes to receive the indicated tracks.

The format of SUBSCRIBE is as follows:

```
SUBSCRIBE Message {
  Track Count (i),
  Track IDs (..),
}
```

Figure 6: Warp SUBSCRIBE Message

* Track Count: The number of track IDs that follow. This **MAY** be zero to unsubscribe to all tracks.

* Track IDs: A list of varint track IDs.

Only the most recent SUBSCRIBE message is active. SUBSCRIBE messages **MUST** be sent on the same QUIC stream to preserve ordering.

6.5. 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 5.7).

* **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 objects from both in parallel.

7. SETUP Parameters

The SETUP message (Section 6.1) allows the peers to exchange arbitrary parameters before any media is exchanged. It is the main extensibility mechanism of Warp. The peers **MUST** ignore unknown parameters. TODO: describe GREASE for those.

Every parameter **MUST** appear at most once within the SETUP message. The peers **SHOULD** verify that and close the connection if a parameter appears more than once.

The ROLE parameter is mandatory for the client. All of the other parameters are optional.

7.1. ROLE parameter

The ROLE parameter (key 0x00) allows the client to specify what roles it expects the parties to have in the Warp connection. It has three possible values:

0x01: Only the client is expected to send media on the connection. This is commonly referred to as *the ingestion case*.

0x02: Only the server is expected to send media on the connection. This is commonly referred to as *the delivery case*.

0x03: Both the client and the server are expected to send media.

The client **MUST** send a ROLE parameter with one of the three values specified above. The server **MUST** close the connection if the ROLE parameter is missing, is not one of the three above-specified values, or it is different from what the server expects based on the application in question.

8. Containers

The container format describes how the underlying codec bitstream is encoded. This includes timestamps, metadata, and generally anything required to decode and display the media.

This draft currently specifies only a single container format. Future drafts and extensions may specifiy additional formats.

ID	Container Format
0x0	fMP4 (Section 8.1)

Table 3

8.1. fMP4

A fragmented MP4 container [ISOBMFF].

The "Container Init Payload" in a CATALOG message (Section 6.3) **MUST** consist 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. A Common Media Application Format Header [CMAF] meets all these requirements.

The "Object Payload" in an OBJECT message (Section 6.2) **MUST** consist of 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. 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.

9. Security Considerations

9.1. 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.

10. IANA Considerations

TODO: fill out currently missing registries: * Warp version numbers * SETUP parameters * Track format numbers * Message types * Object headers

11. 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.

11.1. 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 object **MUST** contain a single track.

11.2. 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 objects with the same configuration.

11.3. 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.

11.3.1. 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:

```
    B     B         B     B         B
   / \   / \       / \   / \       / \
  v   v v   v     v   v v   v     v   v
 I <-- P <-- P   I <-- P <-- P   I <-- P ...
```

11.3.2. 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.

11.3.3. 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.

```
        GoP               GoP            GoP
+-----------------+-----------------+---------------
|     B     B     |     B     B     |     B
|    / \   / \    |    / \   / \    |    / \
|   v   v v   v   |   v   v v   v   |   v   v
|  I <-- P <-- P  |  I <-- P <-- P  |  I <-- P ...
+-----------------+-----------------+--------------
```

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

11.3.4. 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:

```
      +------------------------+-----------------
   4k |  P <- P <- P <- P <- P  |  P <- P <- P ...
      |  |    |    |    |    |  |  |    |    |
      |  v    v    v    v    v  |  v    v    v
      +------------------------+-----------------
1080p |  P <- P <- P <- P <- P  |  P <- P <- P ...
      |  |    |    |    |    |  |  |    |    |
      |  v    v    v    v    v  |  v    v    v
      +------------------------+-----------------
 360p |  I <- P <- P <- P <- P  |  I <- P <- P ...
      +------------------------+-----------------
```

11.4. 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 S ...
```

12. Appendix B. Object Examples

Warp offers a large degree of flexibility on how objects 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.

12.1. Video

12.1.1. Group of Pictures

A group of pictures (GoP) is consists of an I-frame and all frames that directly or indirectly reference it (Section 11.3.3). 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 object consist of a single GoP. For example:

```
     object 1        object 2     object 3
+---------------+---------------+---------
| I  P  B  P  B | I  P  B  P  B | I  P  B
+---------------+---------------+---------
```

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

12.1.2. Scalable Video Coding

Some codecs support scalable video coding (SVC), in which the encoder produces multiple bitstreams in a hierarchy (Section 11.3.4).

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

```
                object 3             object 6
      +-------------------------+---------------
   4k |  P <- P <- P <- P <- P  |  P <- P <- P
      |  |    |    |    |    |  |  |    |    |
      |  v    v    v    v    v  |  v    v    v
      +-------------------------+--------------

                object 2             object 5
      +-------------------------+---------------
1080p |  P <- P <- P <- P <- P  |  P <- P <- P
      |  |    |    |    |    |  |  |    |    |
      |  v    v    v    v    v  |  v    v    v
      +-------------------------+--------------

                object 1             object 4
      +-------------------------+---------------
 360p |  I <- P <- P <- P <- P  |  I <- P <- P
      +-------------------------+---------------
```

12.1.3. Frames

With full knowledge of the encoding, the encoder **MAY** can split a GoP into multiple objects 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 11.3.1) into 13 objects:

```
      2     4           7     9           12
+--------+--------+--------+--------+-----------+
|     B  |  B     |     B  |  B     |     B     |
|-----+--+--+-----+-----+--+--+-----+-----+-----+
|  I  |  P  |  P  |  I  |  P  |  P  |  I  |  P  |
+-----+-----+-----+-----+-----+-----+-----+-----+
   1     3     5     6     8     10    11    13
```

Objects can be merged with their dependency to reduce the total number of objects. QUIC streams will deliver each object in order so the QUIC library performs the reordering.

The same GoP structure can be represented using eight objects:

```
      2     3           5     6           8
+--------+--------+-----------------+------------
|     B  |  B     |     B  |  B     |     B     |
+--------+--------+--------+--------+-----------+
|  I     P     P  |  I     P     P  |  I     P
+-----------------+-----------------+------------
         1                 4              7
```

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

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

```
    object 2      object 4    object 6
+-------------+-------------+---------
|    B   B    |    B   B    |    B
+-------------+-------------+---------
|  I   P   P  |  I   P   P  |  I   P
+-------------+-------------+---------
    object 1      object 3    object 5
```

12.1.4. Init

Initialization data ([Section 11.2](#)) is required to initialize the decoder. Each object **MAY** start with initialization data although this adds overhead.

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

```
     object 2        object 3     object 5
+---------------+---------------+---------
| I  P  B  P  B | I  P  B  P  B | I  P  B
+---------------+---------------+---------
|              init             |  init
+-------------------------------+---------
              object 1            object 4
```

12.2. Audio

Audio ([Section 11.4](#)) is much simpler than video so there's fewer options.

The simplest configuration is to use a single object 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.

```
          object 1
+--------------------------
| S S S S S S S S S S S S S
+--------------------------
```

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

```
     object 1        object 2     object 3
+---------------+---------------+---------
| S   S   S   S   S | S   S   S   S   S | S   S   S
+---------------+---------------+---------
```

This frequency of audio objects is configurable, at the cost of additional overhead. It's **NOT RECOMMENDED** to create a object 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 object at each video I-frame.

```
     object 1        object 3     object 5
+---------------+---------------+---------
| S   S   S   S   S | S   S   S   S   S | S   S   S
+---------------+---------------+---------
| I   P   B   P   B | I   P   B   P   B | I   P   B
+---------------+---------------+---------
     object 2        object 4     object 6
```

12.3. Delivery Order

The delivery order (Section 4.2 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

*Will Law

*Ali Begen

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>.

[RFC9000] 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>.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[WebTransport] Frindell, A., Kinnear, E., and V. Vasiliev, "WebTransport over HTTP/3", Work in Progress, Internet-Draft, draft-ietf-webtrans-http3-05, 13 March 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-webtrans-http3-05>.

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://datatracker.ietf.org/doc/html/draft-cardwell-iccrg-bbr-congestion-control-02>.

[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

Victor Vasiliev
Google

Email: vasilvv@google.com