avtcore WG
Internet-Draft
Intended status: Informational
Expires: 14 September 2026

E. Sprꬽng
Google
G. S. Chandok
Apple
S. Majali
Nvidia

13 March 2026

RTCP Feedback Message and Request Mechanism for Frame-level Acknowledgement
draft-sprang-avtcore-frame-acknowledgement-02

Abstract

This document describes a mechanism for signaling which video frames have been received and decoded by a remote peer. It comprises an RTCP feedback message and an RTP header extension used to request said feedback.

One of the main use cases for this data is to implement various forms of Long Term Reference (LTR) reference structures. Additionally, the mechanism provides a way for receivers to request resynchronization frames that reference acknowledged frames, enabling efficient recovery from partial or full frame loss without requiring a full keyframe.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://github.com/sprangerik/frame-acknowledgement/blob/main/draft-sprang-avtcore-frame-acknowledgement.md. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-sprang-avtcore-frame-acknowledgement/.

Discussion of this document takes place on the avtcore WG Working Group mailing list (mailto:avt@ietf.org), which is archived at https://datatracker.ietf.org/wg/avtcore. Subscribe at https://www.ietf.org/mailman/listinfo/avt/.

Source for this draft and an issue tracker can be found at https://github.com/sprangerik/frame-acknowledgement.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The most common way for realtime video to be transmitted is to encode a pretty much fixed scalability structure, such as those in the W3C [SVC] Scalability mode list.

In such a scenario, the video encoder produces frames "blindly" without real knowledge of what state the remote receiver is in. Using recovery mechanisms such as retransmission, forward error correction and fast-forwarding past skippable frames the receiver is assumed to be able to decode the video. In some cases those methods may not be enough, requiring keyframe requests to be sent as a last resort.

On the other hand, if the encoder is able to reason about which frames have been received and decoded it can be more proactive. One way is to store frames that are known to be received so that they can be later used as guaranteed good references in the case of e.g. large loss events, avoiding the need for potentially large retransmissions etc. Collectively this is often referred to as "Long Term Reference" structures or LTR for short, although the exact structure may vary.

In order to achieve this the sender must be able to reason about the state of the receiver, necessitating the need for feedback signals. In this document a new RTCP message called "Frame Acknowledgement" is introduced as a codec agnostic feedback message for this purpose. Further, an RTP header extension is introduced that allows the sender to actively request feedback on decoding of the associated frame. This allows the sender to both request quick feedback on frames that are important for latency, and enables resilience against loss of feedback packets.

Additionally, there are situations where the receiver may experience partial or full frame loss that cannot be recovered through retransmission or other means. In such cases, the receiver may wish to skip the unrecoverable frame and move forward, but needs a frame encoded with a reference that has been acknowledged. The Frame Acknowledgement Feedback message provides a mechanism for the receiver to request such resynchronization frames, avoiding the need for a full keyframe and thereby minimizing recovery latency and bandwidth usage.

Note that it is allowed to report a frame as decoded even if the decode process is not complete - as long as the receiver guarantees that it will attempt to decode the frame. The rationale for this is that we want to reduce the feedback delay as much as possible. Should the decoding of a frame that has been acknowledged fail, then the receiver MUST request a keyframe to recover, even if the failed decoding belongs to a droppable layer.

2. Conventions and Definitions

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

For the purposes of this document, a "frame" is defined as any decodable unit of bitstream data that results in the update to the codec state (e.g. reference buffers, entropy tables, etc) that can be used as a reference for any subsequent decodable unit of bitstream data. Typically that will be a full frame, but also include cases such as a "no show" frame intended for later reference or even a part of a frame such as a tile or a slice if it is independently decodable and makes updates to encoder state that other tiles/frames can later reference.

3. Applicability

Frame Acknowledgement can be used for video streams in most topologies. It is also designed to be codec agnostic.

In terms of [RFC7667], Point-to-Point is the most straightforward target as it is easiest to reason about a single receiver. A Media Translator or other systems that include a decoder are similarly easy - from the perspective of the sender the middle box is the receiver.

If a Transport Translator is used for Point-to-Multi-Point, then the middlebox must make sure to make valid translations. See section on Frame ID considerations (Section 8) below.

4. Existing Feedback Formats

This section provides an overview, for informational purposes, of some existing feedback formats that could be seen as alternatives.

NACK, defined in [RFC4585], provides only requests for packets the receiver is interested in having retransmitted. Absence of feedback is a poor signal for acknowledgement, especially since said feedback can be lost.

[RFC8888] and [TWCC] provide per-packet acknowledgement and so are more useful. A mapping from packet(s) to frame needs to happen but that is not a big problem. However, even if a frame is confirmed to be received there is no guarantee that it gets decoded.

Reference Picture Selection Indication (RPSI) is another existing message, but it puts the logic of requesting a particular reference frame in the receiver - significantly complicating the system especially in Point-to-Multi-Point systems. It is further codec specific, and several modern codecs lack a specification - including AV1 and H.266.

Loss Notification [LNTF] was a proposed RTCP message intended to solve most of these problems, but it lacks resilience against loss of feedback and also cannot handle out-of-order acknowledgements. The latter makes for instance single-SSRC simulcast structures (e.g. SxTx modes in [SVC]) impossible.

5. Requirements

The messages in this proposal are intended to fulfill the following requirements:

1. Codec agnostic The protocol should be general enough to work across all current and future codecs.

2. Payload Invariant The protocol should not depend on data within the encoded bitstream payload. That includes codec specific frame identifiers, feedback requests and feedback messages.

3. Uses Frame Identifiers Explicit marking of frames, rather than using an indirection via packets.

4. Order Invariant The format should not make assumptions about the required decode order of frames.

5. Send-side Controlled The sender explicitly indicates when and for which frames feedback should be sent.

6. Loss Resilient The sender should be able to detect and recover from lost feedback messages.

7. Low Delay The latency should be small, with the sender being able to tune delay vs rate tradeoff.

8. Low Overhead The network overhead in terms of both packet rate and bitrate should be minimized.

9. Resynchronization Support The receiver should be able to request frames encoded with previously acknowledged references when the decoder state becomes out of sync, enabling efficient recovery without requiring a full keyframe.

6. Frame Acknowledgment Extension

The Frame Acknowledgement extension is an RTP header extension used both to identify frames and request feedback about the remote state. It SHOULD appear on the last packet of a video frame, and MUST NOT appear more than once on a single frame.

6.1. Frame Identifier

In order to request and receive information about decoded frames, we must be able to identify them. The frame acknowledgement header extension may contain a Frame ID field for this purpose. The Frame ID is an 16-bit unsigned integer field, that wraps around to 0 on overflow.

6.2. Frame Acknowledgment Request

In order to get feedback on the state of the remote decoder, the sender actively requests such feedback using the same frame acknowledgement header extension that is also used for frame identification. The feedback request comprises a Start Frame ID and Length field. Specifying the range explicitly has several advantages, including enabling reliable delivery of the feedback since the sender can effectively make retransmission requests of the feedback.

If a new Frame Acknowledgement Request is sent with an incremented Feedback Start, all status values prior to that Frame ID are considered as acknowledged and can be culled by the receiver. A sender MUST NOT request feedback prior to either the last acknowledged Frame ID or the start of the stream.

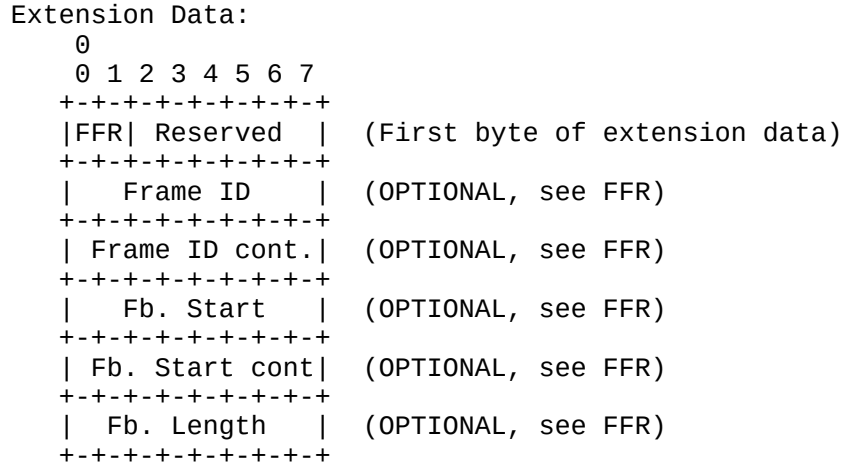
6.3. Frame Acknowledgment Request Data Layout

This section describes the data layout for the Frame Acknowledgment RTP Header Extension. The extension data starts with the FFR/ Reserved byte.

For a One-Byte Header (as defined in [RFC5285], Section 4.2), the ID field identifies the extension, and the len field is a 4-bit value N that indicates the number of data bytes following the ID and len fields, _minus one_. Thus, the total number of bytes for the extension data is N+1.

For a Two-Byte Header (as defined in [RFC5285], Section 4.3), the ID field identifies the extension, and the 8-bit length field indicates the total number of bytes for the extension data (i.e., the FFR/ Reserved byte plus any optional fields).

Extension Data:



6.3.1. FFR: FrameID / Feedback Request (2 bits)

This field is located in the first byte of the extension data. It indicates the presence and meaning of the subsequent optional fields. The total number of bytes for the extension data (and thus the value of N for the one-byte header's len field, or the length field for two-byte headers) depends on the FFR value:

* *00: Frame ID only.* The FFR/Reserved byte is followed by a one-byte Frame ID field. Total extension data bytes = 3 (FFR/Reserved + Frame ID). For one-byte header: len = 2. For two-byte header: length = 3. No feedback is explicitly requested by this header.

* *01: Frame ID + implicit feedback request.* The FFR/Reserved byte is followed by a one-byte Frame ID field. Total extension data bytes = 3 (FFR/Reserved + Frame ID). For one-byte header: len = 2. For two-byte header: length = 3. Feedback is requested for the frame identified by Frame ID (i.e., Feedback Start = Frame ID, Feedback Length = 1).

* *10: Frame ID + independent feedback request.* The FFR/Reserved byte is followed by five bytes: a two-byte Frame ID, a two-byte Feedback Start, and a one-byte Feedback Length field. Total extension data bytes = 6 (FFR/Reserved + Frame ID + Feedback Start + Feedback Length). For one-byte header: len = 5. For two-byte header: length = 6. This implies both a frame marking with Frame ID and an independent feedback request for the specified range.

* *11: Reserved for future use.*

The remaining 6 bits of the FFR/Reserved byte are reserved and SHOULD be set to 0.

6.3.2. Frame ID (16 bits)

Present if FFR is 00, 01, or 10. An unsigned integer that uniquely identifies a frame. It MUST be incremented by one for each new frame (in sending order) that needs to be identified. It wraps around to 0 on overflow.

6.3.3. Feedback Start (16 bits)

Present if FFR is 10. An unsigned integer that corresponds to the first Frame ID (inclusive) the sender is requesting feedback for. It wraps around to 0 on overflow.

6.3.4. Feedback Length (8 bits)

Present if FFR is 10. An unsigned integer that indicates the number of consecutive frames the sender is requesting feedback for, starting from Feedback Start. A value of 0 means no frames are being requested. A value of 1 means only the frame identified by Feedback Start is requested. The range is Feedback Start to Feedback Start + Feedback Length - 1, inclusive, with wrap-around logic applied to Frame IDs.

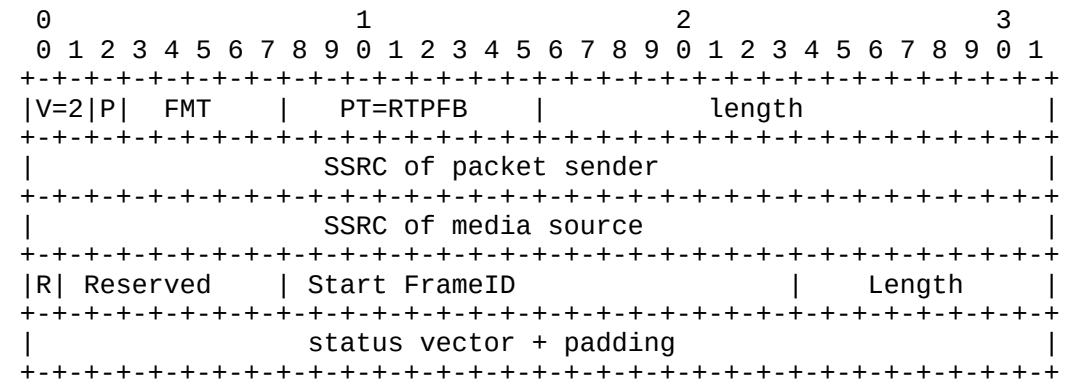
Note that since the Frame ID and Feedback Start are 16-bit fields that wrap, care must be taken when calculating ranges. For example, a request with Feedback Start = 65534 and Feedback Length = 3 indicates the sender is requesting feedback for frames with Frame IDs 65534, 65535, and 0.

If a sender is not interested in feedback for frames prior to and including a given Frame ID, it can effectively signal this by sending a request (FFR=01, 10, or 10) where the Feedback Start (or Frame ID for FFR=01) is more recent. This implicitly acknowledges prior frames up to the new Feedback Start. Alternatively, a Feedback Length of 0 can be used with FFR=10 if no specific frames need feedback but an acknowledgment point needs to be set.

7. Frame Acknowledgment Feedback RTCP Message

The Frame Acknowledgement Feedback message is an RTCP message ([RFC4585]) containing a vector of status symbols, corresponding to the state for the frames requested in a Frame Acknowledgement Extension.

This message is identified by PT = RTPFB (205) and FMT = TBD (to be assigned by IANA, suggested value 12).



7.1. Flags (8 bits)

The flags byte contains the Resync Request Flag and reserved bits for future use.

7.1.1. Resync Request Flag (1 bit)

The most significant bit (bit 0) of the flags byte indicates whether the receiver is requesting a resync frame. When set to 1, indicates that the receiver is requesting a resync frame. When set to 0, acknowledgement is triggered by sender request. If R=1, Start Frame ID should indicate latest decoded frame ID and status vector contatining frames upto latest received Frame ID assuming length field is less than 256.

7.1.2. Reserved (7 bits)

The remaining 7 bits (bits 1-7) are reserved for future use and MUST be set to 0 by senders and MUST be ignored by receivers.

7.2. Start Frame ID (16 bits)

The first Frame ID (inclusive) for which feedback is provided in this message. This corresponds to a Frame ID previously sent in a Frame Acknowledgment Request extension.

7.3. Length (8 bits)

An unsigned integer denoting how many consecutive frames, starting from Start Frame ID, this message contains feedback for. The last Frame ID included in the feedback is (Start Frame ID + Length - 1), with wrap-around logic applied to Frame IDs. A Length of 0 indicates no feedback information is present, though this SHOULD NOT be sent.

7.4. status vector (variable length)

A bit vector of the size indicated by the Length field. Each bit corresponds to a Frame ID, starting from Start Frame ID and incrementing by one for each subsequent bit. * A value of *0* indicates the frame has not been received or has not been decoded (or is not expected to be decoded). * A value of *1* indicates the frame has been received and has been or will be decoded.

The status vector MUST be padded with 0 to align to the next 32-bit boundary if its length is not a multiple of 32 bits. This padding is not included in the Length field but is included in the RTCP packet’s length field.

8. Frame ID considerations

As stated above, the sender MUST increment the Frame ID by one for each new frame with the Frame Acknowledgement header extension present, in sending order. More than that, it must make sure that no wrap-around ambiguity can occur. Since feedback is only really necessary for frames which the codec stores in a reference buffer pending future use, the number of outstanding frames is in practice limited by the number of available reference buffers. E.g. for AV1, the upper limit will be 8. Although the optimal behavior will be application dependent, it is often advisable to spread reference buffer usage out across an RTT and to cull earlier buffer usage once later frames have been acknowledged.

8.1. Resync Request Handling

When a receiver detects that its decoder state has become out of sync with the encoder (for example, due to an unrecoverable partial frame loss), it MAY send a Frame Acknowledgement Feedback message with the R flag (bit 0) set to 1 and specify status vector from latest decoded FrameID upto latest received FrameID.

Upon receiving a resync request, the sender SHOULD: 1. Verify that the decoded Frame ID corresponds to a frame that is still available in its reference buffer. 2. Encode the next frame using the specified frame or another frame with references it knows to be

available at the receiver . 3. If the specified frame is no longer available in the reference buffer, the sender SHOULD encode a keyframe.

This mechanism allows for efficient recovery from decoder desynchronization without the overhead of a full keyframe, as the sender can encode a frame referencing a known good state at the receiver.

8.2. Point-to-Multi-Point

When considering a multi-way application with an SFU/SFM-type relay in the middle, the middlebox may need to do translations/rewriting of Frame IDs such that the outgoing FrameIDs from a middlebox to a receiver still fulfill the requirement that the FrameIDs are incremented by one for each new frame that is marked for feedback. This must be true even if independent video streams for different senders are multiplexed onto the same SSRC. Further the middlebox should typically not acknowledge a frame to a sender unless all active receivers have acknowledged that frame.

8.3. Out-of-order Message Handling

Though rare, it is possible that Frame Acknowledgement Request header extensions are received out of order. This can happen due to e.g., network reordering, but more likely due to retransmissions or recovery of packets using FEC. Regardless of the cause, if a Frame ID is present, the receiver must store it and the state associated with the frame in this packet. If a feedback request is contained in the header extension, and no feedback request has been processed with a Frame ID larger than contained in the requested range, the receiver must process the request. Otherwise, the request must be ignored.

9. SDP Signaling

This section defines how to signal the Frame Acknowledgement RTP header extension and the Frame Acknowledgement Feedback RTCP message using the Session Description Protocol (SDP).

9.1. RTP Header Extension

The Frame Acknowledgement extension is declared in SDP using the "extmap" attribute. The extension does not use any extension attributes.

The URI for declaring this header extension in an extmap attribute is "urn:ietf:params:rtp-hdrext:frame-acknowledgement".

Example attribute line in SDP:

```
a=extmap:4 urn:ietf:params:rtp-hdrext:frame-acknowledgement
```

The extension identifier (4 in the example) is chosen per [RFC8285] and MUST be unique within the media description.

9.2. RTCP Feedback

Support for the Frame Acknowledgement Feedback RTCP message is signaled using the "rtcp-fb" attribute as defined in [RFC4585]. The feedback type "frame-acknowledgement" indicates that the endpoint supports sending and/or receiving the Frame Acknowledgement Feedback message (PT=RTPFB, FMT as assigned by IANA for this feedback type).

The "rtcp-fb" attribute is specified with a payload type value that identifies the RTP payload format for which Frame Acknowledgement Feedback is supported.

Syntax:

```
a=rtcp-fb:<payload type> frame-acknowledgement
```

When used in an offer/answer context, inclusion of "a=rtcp-fb:96 frame-acknowledgement" (with the appropriate payload type for the media) in the SDP indicates that the sender of the SDP is capable of receiving Frame Acknowledgement Feedback messages for the indicated payload type, and that the receiver of the SDP may send Frame Acknowledgement Feedback messages when the RTP header extension is also negotiated for the same media.

9.3. Receiver-Triggered Resync

A receiver that supports sending resync requests (R=1 in the Frame Acknowledgement Feedback message) MAY indicate that it will trigger resync based on decode starvation, and MAY configure the timeout for doing so, using an optional parameter on the "frame-acknowledgement" rtcp-fb attribute.

The "resync-timeout" parameter specifies the time in milliseconds that the receiver will wait for decoding to make progress before sending a resync request. Decode starvation occurs when the receiver cannot advance decoding (e.g., it is blocked waiting for a frame or data that cannot be recovered). If decoding does not make progress for the specified duration, the receiver should send a Frame Acknowledgement Feedback message with the R flag set and a Resync Frame ID referencing the last successfully decoded frame.

Syntax:

a=rtcp-fb:<payload type> frame-acknowledgement;resync-timeout=<timeout-ms>

The value "timeout-ms" is an integer in the range 1-65535, representing the timeout in milliseconds. If "resync-timeout" is omitted, the receiver MAY still send resync requests at its discretion (e.g., on unrecoverable loss) but need not use a timeout-based trigger. Inclusion of "resync-timeout" indicates that the receiver supports and shall use timeout-based resync when decode starves for at least the given duration.

Example attribute lines in SDP (Only one of the format must be present per payload type):

```
a=rtcp-fb:96 frame-acknowledgement
Or
a=rtcp-fb:96 frame-acknowledgement;resync-timeout=500
```

The first format signals support only for Frame Acknowledgement Feedback. The second format additionally signals that the receiver shall trigger resync after 500 ms of decode starvation. "resync-timeout" helps use-cases to choose how long receiver need to wait before triggering resync request. Media sender can adjust its interval to request frame acknowledgement feedback if "resync-timeout" based resync feedback is supported by receiver. Media sender can avoid sending frequent frame acknowledgement requests depending on the use-case. Additionally sender could consider RTT during handling of resync feedbacks.

10. Security Considerations

The messages in this proposal may expose a small amount of data, namely the number of frames that have been sent, and potentially in an indirect way which frames the sender sees as important for recovery.

This data should however not pose any significant privacy or security risks.

11. IANA Considerations

The RTP header extension needs to have a URI identifier assigned by IANA. See [IANAEXT].

The RTCP message uses PT = 205 (RTPFB, Generic RTP Feedback). As of writing, the next available FMT value is 12. A dedicated ID needs to be assigned by IANA. See [IANARTCP].

12. References

12.1. Normative References

[DD] AOM, "Dependency Descriptor RTP Header Extension", n.d., <https://aomediacodec.github.io/av1-rtp-spec/#dependency-descriptor-rtp-header-extension>.

[IANAEXT] IANA, "RTP Compact Header Extensions", n.d., <https://www.iana.org/assignments/rtp-parameters/rtp-parameters.xhtml#rtp-parameters-10>.

[IANARTCP] IANA, "FMT Values for RTPFB Payload Types", n.d., <https://www.iana.org/assignments/rtp-parameters/rtp-parameters.xhtml#rtp-parameters-4>.

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

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

12.2. Informative References

[LNTF] "RTCP feedback Message for Loss Notification", n.d., <https://www.ietf.org/archive/id/draft-majali-avtcore-lntf-feedback-message-00.html>.

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, DOI 10.17487/RFC4585, July 2006, <https://www.rfc-editor.org/rfc/rfc4585>.

[RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, DOI 10.17487/RFC5285, July 2008, <https://www.rfc-editor.org/rfc/rfc5285>.

[RFC7667] Westerlund, M. and S. Wenger, "RTP Topologies", RFC 7667, DOI 10.17487/RFC7667, November 2015, <https://www.rfc-editor.org/rfc/rfc7667>.

[RFC8285] Singer, D., Desineni, H., and R. Even, Ed., "A General Mechanism for RTP Header Extensions", RFC 8285, DOI 10.17487/RFC8285, October 2017, <https://www.rfc-editor.org/rfc/rfc8285>.

[RFC8888] Sarker, Z., Perkins, C., Singh, V., and M. Ramalho, "RTP Control Protocol (RTCP) Feedback for Congestion Control", RFC 8888, DOI 10.17487/RFC8888, January 2021, <https://www.rfc-editor.org/rfc/rfc8888>.

[SVC] W3C, "Scalable Video Coding (SVC) Extension for WebRTC", n.d., <https://www.w3.org/TR/webrtc-svc>.

[TWCC] "RTP Extensions for Transport-wide Congestion Control", n.d., <https://datatracker.ietf.org/doc/html/draft-holmer-rmcat-transport-wide-cc-extensions-01>.

[VFTI] "Video Frame Tracking Id", n.d., <http://www.webrtc.org/experiments/rtp-hdrext/video-frame-tracking-id>.

Appendix A: Example Flows

Notation Legend

The sequence diagrams in this appendix use the following notation:

* *TS*: Timestamp - the RTP timestamp of the video frame
* *FFR*: FrameID / Feedback Request field in the Frame Acknowledgement extension
* *ID*: Frame ID - unique identifier assigned to frames marked for acknowledgement
* *refs*: References - indicates which frame(s) this frame uses as a reference for decoding. This information is contained in the encoded bitstream.
* *FbStart*: Feedback Start - the first Frame ID being requested in feedback
* *FbLen*: Feedback Length - the number of consecutive Frame IDs being requested
* *R*: Resync Request flag in RTCP feedback (R=0: normal feedback, R=1: resync request)

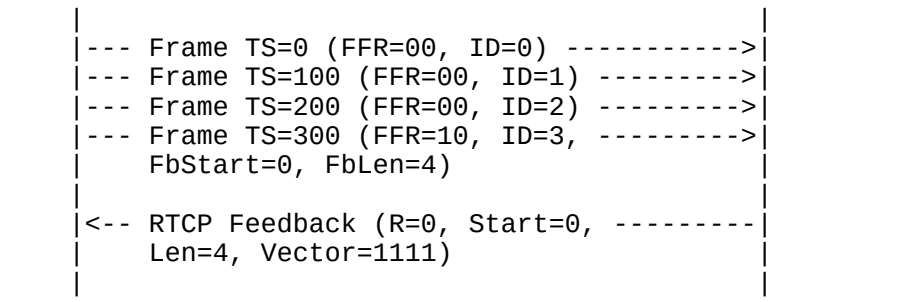
* *Len*: Length field in RTCP feedback - number of Frame IDs included in the status vector

* *Vector*: Status vector in RTCP feedback - bit vector indicating decoded status (1=decoded, 0=not decoded)

Normal Operation Flow

In this scenario, the Media Sender transmits several frames and then requests feedback to confirm which frames have been decoded by the Media Receiver.

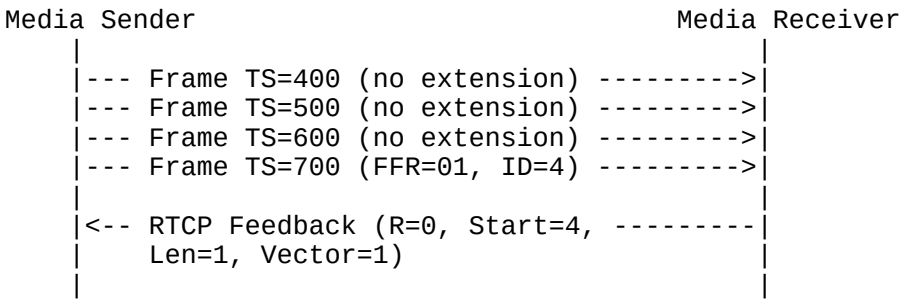
The Media Sender begins by sending frames with Frame IDs but without requesting feedback (FFR=00). On the fourth frame, the sender requests feedback for all frames sent so far using FFR=10 with Feedback Start=0 and Feedback Length=4.



The Media Receiver decodes all four frames and updates its status vector. On decoding the fourth frame it responds with an RTCP Frame Acknowledgement Feedback message with R=0 (not a resync request), Start Frame ID=0, Length=4, and a status vector of 1111 indicating all four frames were successfully decoded.

The Media Sender now knows that Frame IDs 0-3 are confirmed as decoded and can safely be used as long-term references.

For subsequent frames, the sender may choose not to mark some frames with Frame IDs if feedback is not needed. When confirmation is needed for a specific frame, the sender can use implicit feedback requests (FFR=01):



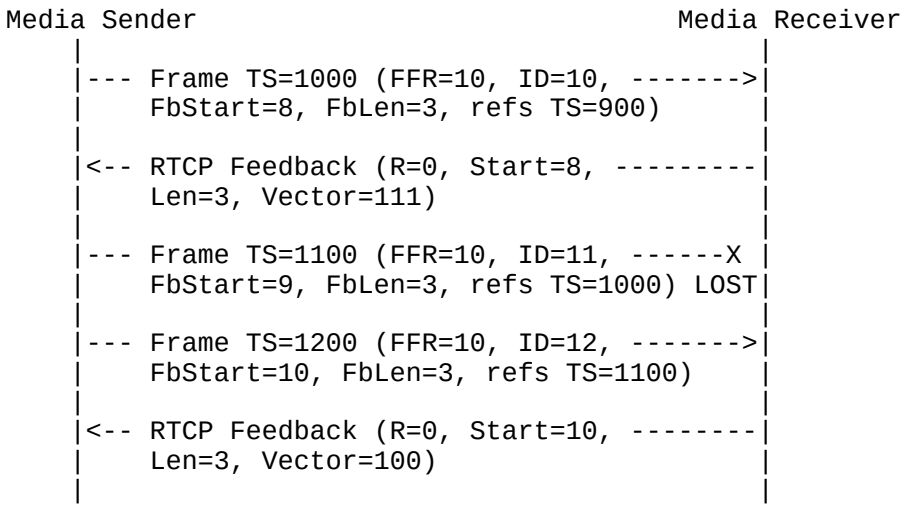
Frames at timestamps 400, 500, and 600 are sent without the Frame Acknowledgement extension since the sender does not need feedback for them. The frame at timestamp 700 is marked with Frame ID=4 (incrementing from the last assigned ID) and uses FFR=01 for an implicit feedback request.

On decoding the frame with ID=4 updates its status vector with just one entry for Frame ID=4. It sends feedback with Start Frame ID=4, and implicitly signals that frames prior to ID=4 (Frame IDs 0-3) are no longer being tracked.

Sender-side Recovery From Frame Loss

In this scenario, a frame is lost in transit. The Media Sender uses a pattern of requesting feedback for the last 2 frames plus the current frame on each feedback request.

The Media Receiver receives the frame with ID=10 and decodes it successfully, but the frame with ID=11 is lost. The frame with ID=12 is received but cannot be decoded because it references the missing frame at timestamp 1100.



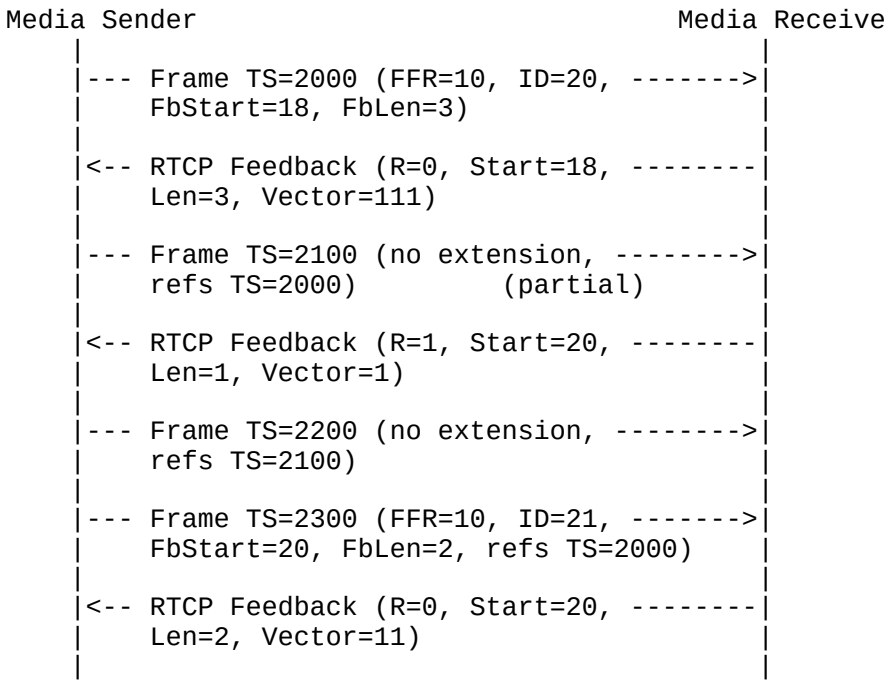
The Media Receiver sends feedback indicating that Frame ID 10 was decoded (bit set to 1), while Frame IDs 11 and 12 were not decoded (bits set to 0). The feedback is sent when it is time to decode the frame with ID=12 to meet its playout deadline, but it is found to have a missing reference (the frame at timestamp 1100).

Upon receiving this feedback, the Media Sender updates its state: Frame ID 10 confirmed decoded, Frame IDs 11 and 12 are marked as not decoded. The sender can choose to encode future frames without referencing the frames at timestamps 1100 or 1200.

Receiver-Triggered Resync Request

Continuing from the frame loss scenario, suppose a frame at timestamp 2100 was partially received. When it is time to decode this frame to meet its presentation deadline, the receiver discovers that it is incomplete and marks it as lost. Since retransmission is not viable within the latency budget and the Media Receiver’s decoder is out of sync, the receiver immediately sends a resync request.

The Media Receiver sends an RTCP Frame Acknowledgement Feedback message with the R flag set to 1, indicating a resync request. The Start Frame ID points to the latest successfully decoded frame (Frame ID 20), and the status vector shows the state of frames from that point up to the latest received Frame ID.



How the sender generates the refresh frame:

Upon receiving the resync request (R=1) with status vector 1, the Media Sender:

1. Examines the feedback to identify the latest decoded Frame ID from the status vector (Frame ID 20 in this case)

2. Checks if the frame at timestamp 2000 (Frame ID 20) is still available in its reference buffer

3. Since the frame at timestamp 2000 is available, encodes the frame at timestamp 2300 using only the frame at timestamp 2000 as a reference

4. The frame at timestamp 2300 is marked with Frame ID=21 (incrementing from the last assigned ID of 20) and uses FFR=10 to request feedback to confirm it was decoded

Frames at timestamps 2100 and 2200 are sent without the Frame Acknowledgement extension since the sender does not need feedback for them. The frame at timestamp 2200 arrives but cannot be decoded since it references the frame at timestamp 2100. The Media Receiver receives the frame at timestamp 2300 (assigned Frame ID=21) and can

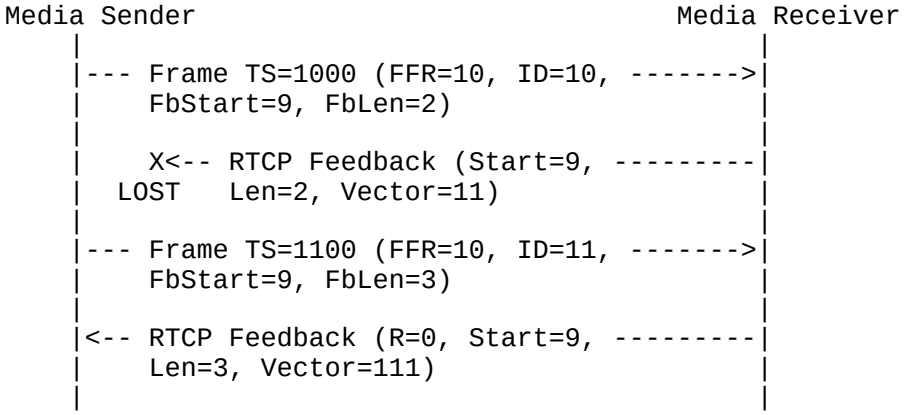
decode it successfully since it only references the frame at timestamp 2000 (Frame ID 20). The decoder is back in sync, and the receiver sends acknowledgement with R=0, Start Frame ID=20, Length=2, and status vector=11 (Frame IDs 20 and 21 decoded).

If the Media Sender no longer had the frame at timestamp 2000 available in its reference buffer, it would instead encode and send a keyframe (IDR frame) to allow the receiver to resynchronize.

Feedback Loss and Recovery

The Frame Acknowledgement mechanism provides resilience against lost feedback messages through re-requests.

In this scenario, the Media Sender requests feedback for frames with IDs 10-11, but the RTCP feedback message from the receiver is lost in transit.



The Media Sender detects that no feedback was received for its earlier request. After a timeout when sending new frames, it re-requests feedback for Frame IDs 9-11 by sending the frame with ID=11 using FFR=10, Feedback Start=9, and Feedback Length=3.

The Media Receiver responds with updated feedback for the requested range, confirming all three Frame IDs (9, 10, 11) have been decoded. The sender now has the confirmation it needed despite the earlier feedback loss.

This mechanism allows the sender to control feedback reliability by re-requesting as needed, providing resilience against both media packet loss and feedback packet loss.

Acknowledgments

Authors' Addresses

Erik Sprꢀng
Google
Email: sprang@google.com

Gurtej Singh Chandok
Apple
Email: gchandok.ietf@gmail.com

Shridhar Majali
Nvidia
Email: smajali@nvidia.com