Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 19, 2018

C. Jennings
Cisco

March 18, 2018

Modular Media Stack
draft-jennings-dispatch-new-media-01

Abstract

A sketch of a proposal for a modular media stack for interactive communications.

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 http://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 September 19, 2018.

Table of Contents

1. Introduction

This draft is an accumulation of varios ideas some people are thinking about. Most of them are fairly separable and could be morphed into existing protocols though this draft takes a blank sheet of paper approach to considering what would be the best think if we were starting from scratch. With that is place, it is possible to ask which of theses ideas makes sense to back patch into existing protocols.

2. Goals

- Better connectivity by enable situation where asymmetric media is possible.
- Design for SFU (Switch Forwarding Units). Design for multiparty calls first then consider two party calls as a specialized subcase of that.
- Designed for client servers with server based controll of clients
- Faster setup
- Pluggable congestion controll
- much much simpler
- end to end security
- remove ability to use STUN / TURN in DDOS reflection attacks
- ability for receiver of video to tell the sender about size changes of display window such that the sender can match
- Eliminiate the problems with ROC in SRTP
- address reasons people have not used from SDES to DTLS-SRTP
- seperation of call setup and ongoing call / conference control

- make codec negotiation more generic so that it works for future codecs

- remove ICE's need for global pacing which is more or less imposible on general purpose devices like PCs

3. Overview

This draft proposes a new media stack to replace the existing stack RTP, DTLS-SRTP, and SDP Offer Answer. The key parts of this stack are connectivity layer, the transport layer, the media layer, a control API, and the singling layer.

The connectivity layer uses a simplified version of ICE, called snowflake [I-D.jennings-dispatch-snowflake], to find connectivity between endpoints and change the connectivity from one address to another as different networks become available or disappear. It is based on ideas from [I-D.jennings-mmusic-ice-fix].

The transport layer uses QUIC to provide a hop by hop encrypted, congestion controlled transport of media. Although QUIC does not currently have all of the partial reliability mechanisms to make this work, this draft assumes that they will be added to QUIC.

The media layer uses existing codecs and packages them along with extra header information to provide information about, when the sequence needs to be played back, which camera it came from, and media streams to be synchronized.

The control API is an abstract API that provides a way for the media stack to report it capabilities and features and a way for the an application tell the media stack how it should be configured. Configuration includes what codec to use, size and frame rate of video, and where to send the media.

The singling layer is based on an advertisement and proposal model. Each endpoint can create an advertisement that describes what it supports including things like supported codecs and maximum bitrates. A proposal can be sent to an endpoint that tells the endpoint exactly what media to send and receive and where to send it. The endpoint can accept or reject this proposal in total but cannot change any part of it.

4. Terminology

- media stream: Stream of information from a single sensor. For example, a video stream from a single camera. A stream may have multiple encodings for example video at different resolutions.

- encoding: A encoded version of a stream. A given stream may have several encodings at different resolutions. One encoding may depend on other encodings such as forward error corrections or in the case of scalable video codecs.

- flow: A logical transport between two computers. Many media streams can be transported over a single flow. The actually IP address and ports used to transport data in the flow may change over time as connectivity changes.

- message: some data or media that to be sent across the network along with metadata about it. Similar to an RTP packet.

- media source: a camera, microphone or other source of data on an endpoint

- media sink: a speaker, screen, or other destination for data on an endpoint

- TLV: Tag Length Value. When used in the draft, the Tag, Length, and any integer values are coded as variable length integers similar to how this is done in CBOR.

5. Architecture

Much of the deployments architecture of IETF media designs are based on a distributed controller for the media stack that is running peer to peer in each client. Nearly all deployments, by they a cloud based conferencing systems or an enterprise PBX, use a central controller that acts as an SBC to try and controll each client. The goal here would be an deployment architecture that

- support a single controller that controlled all the device in a given conference or call. The controller could be in the cloud or running on one of the endpoints.

- design for multi party conference calls first and treat 2 party calls as a specialed sub case of that

- design with the assumption that an light weight SFU (Switched Forwarding Unit) was used to distribute media for conference calls.

6. Connectivity Layer

6.1. Snowflake - New ICE

All that is needed to discover the connectivity is way to:

- Gather some IP/ports that may work using TURN2 relay, STUN2, and local addresses.
- A controller, which might be running in the cloud, to inform a client to send a STUN2 packet from a given source IP/port to a given destination IP/port.
- The receiver notifies the controller about information on received STUN2 packets.
- The controller can tell the sender the secret that was in the packet to prove consent of the receiver to receive data then the sending client can allow media to flow over that connection.

The actually algorithm used to decide on what pairs of addresses are tested and in what order does not need to be agreed on by both the sides of the call - only the controller needs to know it. This allows the controller to use machine learning, past history, and heuristics to find an optimal connection much faster than something like ICE.

The details of this approach are described in [I-D.jennings-dispatch-snowflake]. Many of ideas in this can be traced back to [I-D.kaufman-rtcweb-traversal].

6.2. STUN2

The speed of setting up a new media flow is often determined by how many STUN2 checks need to be done. If the STUN2 packets are smaller, then the stun checks can be done faster without risk of causing congestion.

6.2.1. STUN2 Request

A STUN2 request consists of, well, really nothing. The STUN client just opens a QUIC connection to the STUN server.

6.2.2. STUN2 Response

When the STUN2 sever receives a new QUIC connection, it responds with the IP address and port that the connection came from.

The client can check it is talking to the correct STUN server by checking the fingerprint of the certificate. Protocols like ICE

would need to exchange theses fingerprints instead of all the crazy stun attributes.

Thanks to Peter Thatcher for proposing STUN over QUIC.

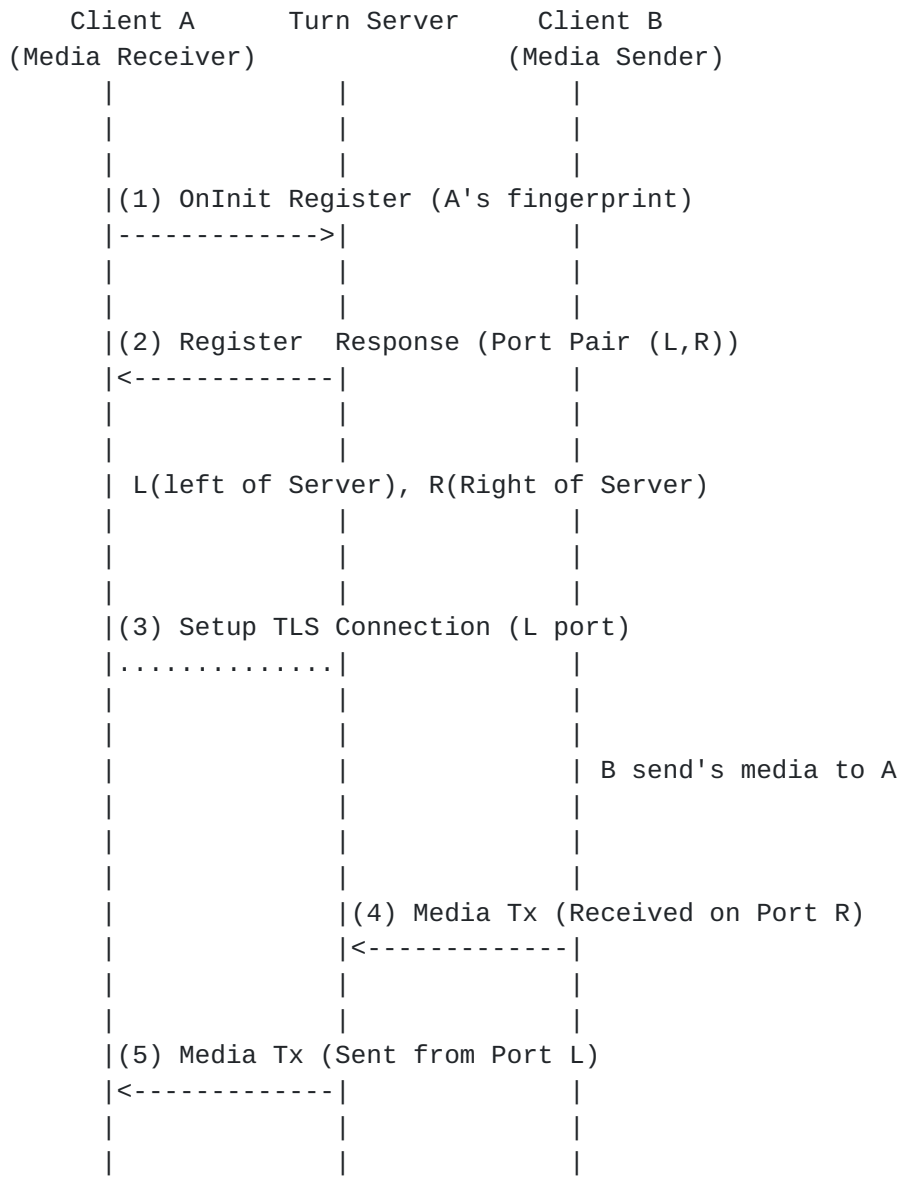
6.3. TURN2

TODO: make TURN2 run over QUIC

Out of band, the client tells the TURN2 server the fingerprint of the cert it uses to authenticate with. The TURN2 server gives the client two public IP:port address pairs. One is called inbound and other called outbound. The client connects to the outbound port and authenticates to TURN2 server using the TLS domain name of server. The TURN2 server authenticates the client using mutual TLS with fingerprint of cert provided by the client. Any time a message or stun packet is received on the matched inbound port, the TURN2 server forwards it to the client(s) connected to the outbound port.

A single TURN2 connection can be used for multiple different calls or session at the same time and a client could choose to allocate the TURN2 connection at the time that it started up. It does not need to be done on a per session basis.

The client can not send from the TURN2 server.



7. Transport Layer

The responsibility of the transport layer is to provide an end to end crypto layer equivalent to DTLS and they must ensure adequate congestion control. The transport layer brings up a flow between two computers. This flow can be used by multiple media streams.

The MTI transport layer is QUIC with packets. It assumes that QUIC has a way to delivers the packets in an effecent unreliable mode as wells as an optional way to deliver important metadata packets in a reliable mode. It assumes that QUIC can report up to the rate adaptation layer a current max target bandwidth that QUIC can transmit at. It's possible these are all unrealistic characteristics

of QUIC in which case a new transport protocol should be developed that provides these and is layered on top of DTLS for security.

This is secured by checking the fingerprints of the DTLS connection match the fingerprints provided at the control layer or by checking the names of the certificates match what was provided at control layer.

The transport layer needs to be able to set the DSCP values in transmitting packets as specified by the control layer.

The transport MAY provide a compression mode to remove the redundancy of the non-encrypted portion of the media messages such as GlobalEncodingID. For example, a GlobalEncodingID could be mapped to a QUIC channel and then it could be removed before sending the message and added back on the receiving side.

The transport need to be able to ensure that it has a very small chance of being confused with the STUN2 traffic it will be multiplexed with. (Open issue - if the STUN2 runs on top of same transport, this becomes less of issue)

The transport crypto needs to be able to export server state that can be passed out of band to the client to enable the client to make a zero RTT connection to the server.

8. Media Layer - RTP3

Each message consist of a set of TLV headers with metadata about the packet, followed by payload data such as the output of audio or video codec.

There are several message headers that help the receiver understand what to do with the media. The TLV header are the follow:

- Conference ID: Integer that will be globally unique identifier for the for all applications using a common call singling system. This is set by the proposal.

- Endpoint ID: Integer to uniquely identify the endpoint with within scope of conference ID. This is set by the proposal.

- Source ID: integer to uniquely identify the input source within the scope a endpoint ID. A source could be a specific camera or a microphone. This is set by the endpoint and included in the advertisement.

- Sink ID: integer to uniquely identify the sink within the scope a endpoint ID. A sink could be a speaker or screen. This is set by the endpoint and included in the advertisement. An endpoint sending media can have this set. If it is set it should transmit it for 3 frames any time it changes and once every 5 second. An SFU can add, modify, or delete this from any media packet. TODO - How to use this for SFU controlled layout - for example, if have 100 users in conference and want to put the 10 most recent speakers in thumbnails. Do we need this at all ?

- Encoding ID: integer to uniquely identify the encoding of the stream within the scope of the source ID. Note there may be multiple encodings of data from the same source. This is set by the proposal.

- Salt : salt to use for forming the initialization vector for AEAD. The salt shall be sent as part of the packet and need not be sent in all the packets. This is created by the endpoint sending the message.

- GlobalEncodingID: 64 bit hash of concatenation of conference ID, endpoint ID, source ID, encoding ID

- Capture time: Time when the first sample in the message was captured. It is a NTP time in ms with the high order bits discarded. The number of bits in the capture time needs to be large enough that it does not wrap in for the lifetime of this stream. This is set by the endpoint sending the message.

- Sequence ID: When the data captured for a single point in time is too large to fit in a single message, it can be split into multiple chunks which are sequentially numbered starting at 0 corresponding to the first chunk of the message. This is set by the endpoint sending the message.

- GlobalMessageID: 64 bit hash of concatenation of conference ID, endpoint ID, encoding ID, sequence ID

- Active level: this is a number from 0 to 100 indicates the probability that the sender of this media wishes it to be considered active media. For example if it was voice, it would be 100 if the person was clearly speaking, and 0 if not, and perhaps a value in the middle if it was uncertain. This allows an media switch to select the active speaker in the in a conference call.

- Location: relative or absolute location, direction of view, and field view. With video coming from drones, 360 cameras, VR light field cameras, and complex video conferencing rooms, this provides

the information about the camera or microphone that the receiver can use to render the correct view. This is end to end encrypted.

- Reference Frame : bool to indicate if this message is part of a reference frame. Typically, a SFU will switch to the new video stream at the start of a reference frame.

- DSCP : DSCP to use on transmissions of this message and future messages on this GlobalEncodingID

- Layer ID : Integer indicating which layer is for scalable video codecs. SFU may use this to selectively drop a frame.

The keys used for the AEAD are unique to a given conference ID and endpoint ID.

If the message has any of the following headers, they must occur in the following order followed by all other headers:

1. GlobalEncodingID,

2. GlobalMessageID,

3. conference ID,

4. endpoint ID,

5. encoding ID,

6. sequence ID,

7. active level,

8. DSCP

Every second there much be at least one message in each encoding that contains:

- conference ID,

- endpoint ID,

- encoding ID,

- salt,

- and sequence ID headers

but they are not needed in every packet.

The sequence ID or GlobalMessageID is required in every message and periodically there should be message with the capture time.

8.1. RTP Meta Data

We tend to end up with a few categories of data associated with the media:

- Stuff you need at the same time you get the media. For example, this is a reference frame.
- Stuff you need soon but not instantly. For example the name of the speaker in a given rectangle of a video stream

And it tends to change at different rates:

- Stuff that you need to process the media and may change but does not change quickly and you don't need it with every frame. For example, salt for encryption
- Stuff that you need to join the media but may never change. For example, resolution of the video is

TODO - think about how to optimize design for each type of meta data

8.2. Securing the messages

The whole message is end to end secured with AEAD. The headers are authenticated while the payload data is authenticated and encrypted. Similar to how the IV for AES-GCM is calculated in SRTP, in this case the IV is computed by xor'ing the salt with the concatenation of the GlobalEncodingID and low 64 bits of sequence ID. The message consists of the authenticated data, followed by the encrypted data , then the authentication tag.

8.3. Sender requests

The control layer supports requesting retransmission of a particular media message identified by IDs and capture time it would contain.

The control layer supports requesting a maximum rate for each given encoding ID.

8.4. Data Codecs

Data messages including raw bytes, xml, senml can all be sent just like media by selecting an appropriate codec and a software based source or sink. An additional parameter to the codec can indicate if reliably delivery is needed and if in order delivery is needed.

8.5. Media Keep Alive

Provided by transport.

8.6. Forward Error Correction

A new Reed-Solomon based FEC scheme based on [I-D.ietf-payload-flexible-fec-scheme] that provides FEC over messages needs to be defined.

8.7. MTI Codecs

8.7.1. Audio

Implementation MUST support at least G711 and Opus

8.7.2. Video

Implementation MUST support at least H.264 and AV1

Video codecs use square pixels.

Video codecs MUST support any aspect ratio within the limits of their max width and height.

Video codecs can specify a maximum pixel rate, maximum frame rate, maximum images size. The can also specify a list of binary flags of supported features which are defined by the codec and may be supported by the codec for encode, decode, or neither where each feature can be independently controlled. They can not impose constraints beyond that. Some existing codecs like vp8 may easily fit into that while some codec like H264 may need some suspects defined as new codecs to meet the requirements for this. It is not expected that all the nuances that could be negotiated with SDP for 264 would be supported in this new media.

Video codecs MUST support a min width and min height of 1.

All video on the wire is oriented such that the first scan line in the frame is up and first pixel in the scan line is on the left.

T.38 fax and DTMF are not supported. Fax can be sent as a TIFF imager over a data channel and DTFM can be done as an application specific information over a data channel.

TODO: Capture the list of what metadata video encoders produce * if it is a reference frame or not * resolution * frame-rate ? * capture time of frame

TODO: Capture the list of what metadata video encoders needs. * capture timestamp * source and target resolution * source and target frame-rate * target bitrate * max bitrate * max pixel rate

8.7.3. Annotation

Optional support for annotation based overlay using vector graphics such as a subset of SVG.

8.7.4. Application Data Channels

Need support for application defined data in both a reliable and unreliable datagram mode.

8.7.5. Reverse Requests & Stats

The hope is that this is not needed.

Much of what goes in the reverse direction of the media in RTCP is either used for congestion controll, diagnostics, or controll of the codec such as requesting to resent a frame or sending a new intra codec frame for video. The design reduces the need for this.

The congestion controll information which is needed quickly is all handled at QUIC layer.

The diagnostic type information can be reported from the endpint to the controller and does not need to flow at the media level.

Information that needs to be delivered reliably can be sent that way at the QUIC level remove the need for retransmit type request. System that use selective retransmission to recover from packet loss of media do not tend to work as well for interactive medias as forward error correction schemes because of the large latency they introduce.

Information like requesting a new intra codec frame for video often needs to come from the controller and can be sent over the signalling and controll layer.

8.8. Message Key Agreement

The secret for encrypting messages can be provided in the proposal by value or by a reference. The reference approach allows the client to get it from a messaging system where the server creating the proposal may not have access to the the secret. For example, it might come from a system like [I-D.barnes-mls-protocol].

9. Control Layer

The control layer needs an API to find out what the capabilities of the device are, and then a way to set up sending and receiving stream. All media flow are only in one direction. The control is broken into control of connectivity and transports, and control of media streams.

9.1. Transport Capabilities API

An API to get information for remote connectivity including:

- set the IP, port, and credential for each TURN2 server
- can return the IP, port tuple for the remote side to send to TURN2 server
- gather local IP, port, protocol tuples for receiving media
- report SHA256 fingerprint of local TLS certificate
- encryption algorithms supported
- report an error for a bad TURN2 credential

9.2. Media Capabilities API

Send and receive codecs are consider separate codecs and can have separate capabilities though the default to the same if not specified separately.

For each send or receive audio codec, an API to learn:

- codec name
- the max sample rate
- the max sample size
- the max bitrate

For each send or receive video codec, an API to learn:

- codec name
- the max width
- the max height
- the max frame rate
- the max pixel depth
- the max bitrate
- the max pixel rate (pixels / second)

9.3. Transport Configuration API

To create a new flow, the information that can be configured is:

- turn server to use
- list of IP, Port, Protocol tuples to try connecting to
- encryption algorithm to use
- TLS fingerprint of far side

An api to allow modification of the follow attributes of a flow:

- total max bandwidth for flow
- forward error correction scheme for flow
- FEC time window
- retransmission scheme for flow
- addition IP, Port, Protocol pairs to send to that may improve connectivity

9.4. Media Configuration API

For all streams:

- set conference ID
- set endpoint ID

- set encoding ID
- salt and secret for AEAD
- flag to pause transition

For each transmitted audio steam, a way to set the:

- audio codec to use
- media source to connect
- max encoded bitrate
- sample rate
- sample size
- number of channels to encode
- packetization time
- process as one of : automatically set, raw, speech, music
- DSCP value to use
- flag to indicating to use constant bit rate
- optionally set a sinkID to periodically include in the media

For each transmitted video stream, a way to set

- video codec to use
- media source to connect to
- max width and max height
- max encoded bitrate
- max pixel rate
- sample rate
- sample size
- process as one of : automatically set, rapidly changing video, fine detail video

- DSCP value to use
- for layered codec, a layer ID and set of layers IDs this depends on
- optionally set a sinkID to periodically include in the media

For each transmitted video stream, a way to tell it to:

- encode the next frame as an intra frame

For each transmitted data stream:

- a way to send a data message and indicate reliable or unreliable transmission

For each received audio stream:

- audio codec to use
- media sink to connect to
- lip sync flag

For each received video stream:

- video codec to use
- media sink to connect to
- lip sync flag

For each received data stream:

- notification of received data messages

Note on lip sync: For any streams that have the lip sync flag set to true, the render attempts to synchronize their play back.

9.5. Transport Metrics

- report gathering state and completion

9.6. Flow Metrics API

For each flow, report:

- report connectivity state

- report bits sent
- report packets lost
- report estimated RTT
- report SHA256 fingerprint for certificate of far side
- current 5 tuple in use

9.7. Stream Metrics API

For sending streams:

- Bits sent
- packets lost

For receiving streams:

- capture time of most recently receives packet
- endpoint ID of more recently received packet
- bits received
- packets lost

For video streams (send & receive):

- current encoded width and height
- current encoded frame rate

10. Call Signalling - JABBER2

Call signalling is out of scope for usages like WebRTC but other usages may want a common REST API they can use.

Call signalling works be having the client connect to a server when it starts up and send its current advertisement and open a web socket or to receive proposals from the server. A client can make a rest call indicating the parties(s) it wishes to connect to and the server will then send proposals to all clients that connect them. The proposal tell each client exactly how to configure it's media stack and MUST be either completely accepted, or completely rejected.

The signalling is based on the the advertisement proposal ideas from [I-D.peterson-sipcore-advprop].

We define one round trip of signalling to be a message going from a client up to a server in the cloud, then down to another client which returns a response along the reverse path. With this definition SIP is takes 1.5 round trips or more if TURN is needed to set up a call while this takes 0.5 round trips.

11. Signalling Examples

11.1. Simple Audio Example

11.1.1. simple audio advertisement

```
{
  "receiveAt":[
    {
      "relay":"2001:db8::10:443",
      "stunSecret":"s8i739dk8",
      "tlsFingerprintSHA256":"1283938"
    },
    {
      "stun":"203.0.113.10:43210",
      "stunSecret":"s8i739dk8",
      "tlsFingerprintSHA256":"1283938"
    },
    {
      "local":"192.168.0.2:443",
      "stunSecret":"s8i739dk8",
      "tlsFingerprintSHA256":"1283938"
    }
  ],
  "sources":[
    {
      "sourceID":1,
      "sourceType":"audio",
      "codecs":[
        {
          "codecName":"opus",
          "maxBitrate":128000
        },
        {
          "codecName":"g711"
        }
      ]
    }
  ],
```

```
                  "sinks":[
                    {
                      "sinkID":1,
                      "sourceType":"audio",
                      "codecs":[
                        {
                          "codecName":"opus",
                          "maxBitrate":256000
                        },
                        {
                          "codecName":"g711"
                        }
                      ]
                    }
                  ]
                }
```

11.1.2. simple audio proposal

```
                {
                  "receiveAt":[
                    {
                      "relay":"2001:db8::10:443",
                      "stunSecret":"s8i739dk8"
                    },
                    {
                      "stun":"203.0.113.10:43210",
                      "stunSecret":"s8i739dk8"
                    },
                    {
                      "local":"192.168.0.10:443",
                      "stunSecret":"s8i739dk8"
                    }
                  ],
                  "sendTo":[
                    {
                      "relay":"2001:db8::20:443",
                      "stunSecret":"20kdiu83kd8",
                      "tlsFingerprintSHA256":"9389739"
                    },
                    {
                      "stun":"203.0.113.20:43210",
                      "stunSecret":"20kdiu83kd8",
                      "tlsFingerprintSHA256":"9389739"
                    },
                    {
                      "local":"192.168.0.20:443",
                      "stunSecret":"20kdiu83kd8",
```

```
                    "tlsFingerprintSHA256":"9389739"
                  }
                ],
                "sendStreams":[
                  {
                    "conferenceID":4638572387,
                    "endpointID":23,
                    "sourceID":1,
                    "encodingID":1,
                    "codecName":"opus",
                    "AEAD":"AES128-GCM",
                    "secret":"xy34",
                    "maxBitrate":24000,
                    "packetTime":20
                  }
                ],
                "receiveStreams":[
                  {
                    "conferenceID":4638572387,
                    "endpointID":23,
                    "sinkID":1,
                    "encodingID":1,
                    "codecName":"opus",
                    "AEAD":"AES128-GCM",
                    "secret":"xy34"
                  }
                ]
              }
```

11.2. Simple Video Example

Advertisement for simple send only camera with no audio

```
{
  "sources":[
    {
      "sourceID":1,
      "sourceType":"video",
      "codecs":[
        {
          "codecName":"av1",
          "maxBitrate":20000000,
          "maxWidth":3840,
          "maxHeight":2160,
          "maxFrameRate":120,
          "maxPixelRate":248832000,
          "maxPixelDepth":8
        }
      ]
    }
  ]
}
```

11.2.1. Proposal sent to camera

```
{
  "sendTo":[
    {
      "relay":"2001:db8::20:443",
      "stunSecret":"20kdiu83kd8",
      "tlsFingerprintSHA256":"9389739"
    }
  ],
  "sendStreams":[
    {
      "conferenceID":0,
      "endpointID":0,
      "sourceID":0,
      "encodingID":0,
      "codecName":"av1",
      "AEAD":"NULL",
      "width":640,
      "height":480,
      "frameRate":30
    }
  ]
}
```

11.3. Simulcast Video Example

Advertisement same as simple camera above but proposal has two streams with different encodingID.

```
{
  "sendTo":[
    {
      "relay":"2001:db8::20:443",
      "stunSecret":"20kdiu83kd8",
      "tlsFingerprintSHA256":"9389739"
    }
  ],
  "sendStreams":[
    {
      "conferenceID":0,
      "endpointID":0,
      "sourceID":0,
      "encodingID":1,
      "codecName":"av1",
      "AEAD":"NULL",
      "width":1920,
      "height":1080,
      "frameRate":30
    },
    {
      "conferenceID":0,
      "endpointID":0,
      "sourceID":0,
      "encodingID":2,
      "codecName":"av1",
      "AEAD":"NULL",
      "width":240,
      "height":240,
      "frameRate":15
    }
  ]
}
```

11.4. FEC Example

11.4.1. Advertisement includes a FEC codec.

```
{
  "sources":[
    {
      "sourceID":1,
      "sourceType":"video",
      "codecs":[
        {
          "codecName":"av1",
          "maxBitrate":20000000,
          "maxWidth":3840,
          "maxHeight":2160,
          "maxFrameRate":120,
          "maxPixelRate":248832000,
          "maxPixelDepth":8
        },
        {
          "codecName":"flex-fec-rs"
        }
      ]
    }
  ]
}
```

11.4.2. Proposal sent to camera

```
{
  "sendTo":[
    {
      "relay":"2001:db8::20:443",
      "stunSecret":"20kdiu83kd8",
      "tlsFingerprintSHA256":"9389739"
    }
  ],
  "sendStreams":[
    {
      "conferenceID":0,
      "endpointID":0,
      "sourceID":0,
      "encodingID":1,
      "codecName":"av1",
      "AEAD":"NULL",
      "width":640,
      "height":480,
      "frameRate":30
    },
    {
      "conferenceID":0,
      "endpointID":0,
      "sourceID":0,
      "encodingID":2,
      "AEAD":"NULL",
      "codecName":"flex-fec-rs",
      "fecRepairWindow":200,
      "fecRepairEncodingIDs":[
        1
      ]
    }
  ]
}
```

12. Switched Forwarding Unit (SFU)

When several clients are in conference call, the SFU can forward packets based on looking at which clients needs a given GlobalEncodingID. By looking at the "active level", the SFU can figure out which endpoints are the active speaker and forward only those. The SFU never changes anything in the message.

12.1. Software Defined Networking

Is it possible to use the packet recycling concepts in SDN to forward a single packet to multiple endpoints? Can the way SDN forwarding would work be adapted to use a SDN router as a SFU?

12.2. Vector Packet Processors

Can we use fast VPP systems like fd.io to create a SFU?

12.3. Information Centric Networking

What changes would be needed to map RTP2 into the prefix and suffix of hICN?

13. Acknowledgements

Thank you for input from: Harald Alvestrand, Espen Berger, Matthew Kaufman, Patrick Linskey, Eric Rescorla, Peter Thatcher, Malcolm Walters Martin Thomson

14. Other Work

rfc7016

draft-kaufman-rtcweb-traversal

Consider using terminology from rfc7656

docs.google.com/presentation/
d/1Sg_1TVCcKJvZ8Egz5oa0CP01TC2rNdv9HVu7W38Y4zA/
edit#slide=id.g29a8672e18_22_120

docs.google.com/presentation/d/1o-
o5jZBLw3Py1OuenzWDkxDG6NigSmLHvGw5KemKWLw/
edit#slide=id.g2f8f4acff1_1_249

cs.chromium.org/chromium/src/third_party/webrtc/common_video/include/
video_frame.h

15. Style of specification

Fundamental driven by experiments. The proposal is to have a high level overview document where we document some of the design - this document could be a start of that. Then write a a spec for each on of the separable protocol parts such as STUN2, TURN2, etc.

The protocol specs would contain a high level overview like you might find on a wikipedia page and the details of the protocol encoding would be provided in an open source reference implementation. The test code for the references implementation helps test the spec. The implementation is not optimized for perfromance but instead is simply trying to clearly illustrate the protocol. Particular version of the draft would be bound to a tagged version of the source code. All the

source code would be under normal IETF IPR rules just like it was included directly in the draft.

16. Informative References

[I-D.barnes-mls-protocol]
Barnes, R., Millican, J., Omara, E., Cohn-Gordon, K., and R. Robert, "The Messaging Layer Security (MLS) Protocol", draft-barnes-mls-protocol-00 (work in progress), February 2018.

[I-D.ietf-payload-flexible-fec-scheme]
Zanaty, M., Singh, V., Begen, A., and G. Mandyam, "RTP Payload Format for Flexible Forward Error Correction (FEC)", draft-ietf-payload-flexible-fec-scheme-06 (work in progress), March 2018.

[I-D.jennings-dispatch-snowflake]
Jennings, C. and S. Nandakumar, "Snowflake - A Lighweight, Asymmetric, Flexible, Receiver Driven Connectivity Establishment", draft-jennings-dispatch-snowflake-01 (work in progress), March 2018.

[I-D.jennings-mmusic-ice-fix]
Jennings, C., "Proposal for Fixing ICE", draft-jennings-mmusic-ice-fix-00 (work in progress), July 2015.

[I-D.kaufman-rtcweb-traversal]
Kaufman, M. and J. Rosenberg, "NAT Traversal Requirements for RTCWEB", draft-kaufman-rtcweb-traversal-00 (work in progress), June 2011.

[I-D.peterson-sipcore-advprop]
Peterson, J. and C. Jennings, "The Advertisement/Proposal Model of Session Description", draft-peterson-sipcore-advprop-01 (work in progress), March 2011.

Author's Address

Cullen Jennings
Cisco

Email: fluffy@iii.ca