WISH@IETF110

This session is being recorded

- **Make sure your video is off.**
- **Mute your microphone unless you are speaking.**
- **Join the session:**
    - **Meetecho (a/v and chat):** `https://meetings.conf.meetecho.com/ietf110/?group=wish&short=&item=1`
    - **Audio (only):** `http://mp3.conf.meetecho.com/ietf110/wish/1.m3u`
    - **Jabber (chat):** `https://codimd.ietf.org/notes-ietf-110-wish`

I E T F®

Note Well

This is a reminder of IETF policies in effect on various topics such as patents or code of conduct. It is only meant to point you in the right direction. Exceptions may apply. The IETF's patent policy and the definition of an IETF "contribution" and "participation" are set forth in BCP 79; please read it carefully.

As a reminder:

- By participating in the IETF, you agree to follow IETF processes and policies.
- If you are aware that any IETF contribution is covered by patents or patent applications that are owned or controlled by you or your sponsor, you must disclose that fact, or not participate in the discussion.
- As a participant in or attendee to any IETF activity you acknowledge that written, audio, video, and photographic records of meetings may be made public.
- Personal information that you provide to IETF will be handled in accordance with the IETF Privacy Statement.
- As a participant or attendee, you agree to work respectfully with other participants; please contact the ombudsteam (https://www.ietf.org/contact/ombudsteam/) if you have questions or concerns about this.

Definitive information is in the documents listed below and other IETF BCPs. For advice, please talk to WG chairs or ADs:

- BCP 9 (Internet Standards Process)
- BCP 25 (Working Group processes)
- BCP 25 (Anti-Harassment Procedures)
- BCP 54 (Code of Conduct)
- BCP 78 (Copyright)
- BCP 79 (Patents, Participation)
- https://www.ietf.org/privacy-policy/(Privacy Policy)

WISH@IETF110

Tuesday March 9th, Session I
Chairs: Alex Gouaillard, Sean Turner

Agenda

Administrivia (10 mn)

- Virtual Meeting Tips
- Note Well
- Virtual Bluesheet (automatic)
- Note Taker
- Jabber Scribe
- Status (just chartered)

(1) Chairs (10 mn)

- what's in/out of scope ()
- Actions Items from charter discussion

(2) Proposals (40~50 mn)

- Sergio M. (15~20 mn): Original Proposal
- Adam R. (20 mn): Extensions proposal
- Lorenzo M. (5~10 mn)
- Others?

(3) Discussion (40 mn)

Insert graphic here

Chairs Intro: Scope

The WISH working group is chartered to specify a

- simple,
- extensible,
- HTTPS-based
- signaling protocol.

Goal: to establish

- one-way
- WebRTC-based audiovisual sessions
- between broadcasting tools and real-time media broadcast networks.

Chairs: A.I. from Dispatch and Charter discussions

The need for extensibility is something that Jonathan Lennox highlighted during the WHIP discussion at DISPATCH 109

WHIP: Original Proposal

Sergio Garcia Murillo
<sergio.garcia.murillo@gmail.com>

The Problem

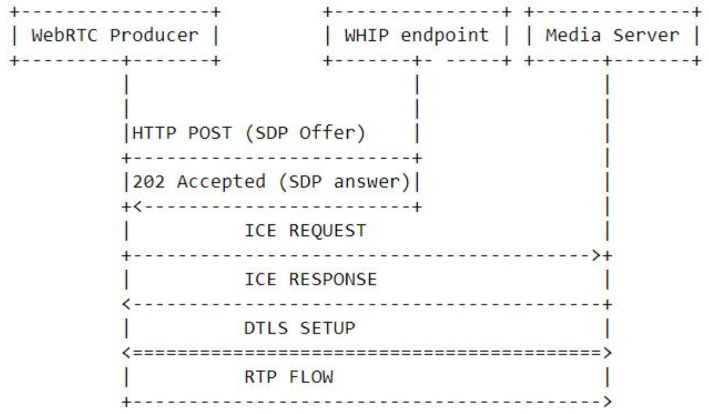
- WebRTC is the best media transport protocol for real-time streaming.
- While other media transport could be used for ingest, webrtc for both ingest and delivery allows:
  - Working on browsers.
  - Avoiding protocol translation, which could add delay and adds implementation complexity.
  - Avoiding transcoding by sharing common codecs.
  - Using webrtc features end to end.
- However, there is no standard and each WebRTC streaming services requires implementing a custom ad-hoc protocol.

Requirements

- Must be simple to implement, as easy to use as current RTMP URI scheme
- Support the specific ingest use case, which is a subset of webrtc possible use cases:
  - Only needs to support unidirectional flows.
  - *Server is assumed to not be behind NAT (having a public IP or deployed in same private network as publisher)*
  - *No need to support renegotiations.*
  - Fully compliant with WebRTC and RTCWEB specs for the given use case.
  - Must support authentication.
  - Usable both in web browsers and in native encoders.
- **Lower the requirements on both hardware encoders and broadcasting by reducing optionalities.**
- Supports load balancing and redirections.

Proposed solution

- HTTP POST for exchanging and SDP O/A.
- Connection state is controlled by ICE/DTLS states
  - ICE consent freshness [RFC7675] will be used to detect abrupt disconnection
  - DTLS teardown for session termination by either side.
- Authentication and authorization is supported by the Authorization HTTP header with a bearer token as per [RFC6750].
- Support HTTP redirections for LB.



WHIP session setup

Example implementation in JS

```js
//Get user media
const stream = await
navigator.mediaDevices.getUserMedia({audio:true,
video:true});
//Create peer connection
const pc = new RTCPeerConnection();
//Listen for state change events
pc.onconnectionstatechange = (event) =>{
          switch(pc.connectionState) {
                    case "connected":
                              break;
                    case "disconnected":
                              break;
                    case "failed":
                              break;
                    case "closed":
                              break;
          }
}
```

```js
//Send all tracks
for (const track of stream.getTracks())
       //You could add simulcast too here
          pc.addTrack(track);
//Create SDP offer
const offer = await pc.createOffer();
await pc.setLocalDescription(offer)
//Do the post request to the WHIP endpoint with the SDP offer
const fetched = await fetch(url, {
          method: "POST",
          body: offer.sdp,
          headers:{
                    "Content-Type": "application/sdp"
          }
});
//Get the SDP answer
const answer = await fetched.text();
await pc.setRemoteDescription({type:"answer",sdp: answer});
```

Reducing implementation complexity

- Server may implement ICE lite, encoder must implement full ICE.
- SDP bundle and RTCP muxing must be supported by both sides.
- Encoder/media producer may use a setup attribute value of setup:active in the SDP offer, server must support acting as passive.

WHIP: Proposed Enhancements

Adam Roach <adam@nostrum.com>

(Potential) Requirements

- Session Management (termination, in particular)
- Extensibility (ability to add new operations and new parameters)
    - This was highlighted during the DISPATCH discussion
- Heartbeats
- Two-way Communication (e.g., ability for ingest gateway to initiate operations towards broadcasting tool)
- ICE management (trickling and restarts)
- Some way of knowing which media types (and how many) will be sent.

General Solution Constraints

- Since we are using HTTP as our basis, there are two key documents we need to stay aware of:
  - **BCP 56**, which defines how applications can use HTTP as a substrate.
  - **BCP 190**, which defines how applications use HTTP URLs.
- The most notable restrictions are:
  - No new HTTP methods/headers/responses unless they impact HTTP processing.
  - Use existing methods/headers/responses only to mean what HTTP defines them to mean.
  - Do not define rules for building HTTP URLs (e.g., adding components to a base path).

Session Management Options

- Option 1: Send POST to session creation URL, and use HTTP “201 Created” response with “Location” header field pointing to a newly-created URL, representing the session.
  - Suboption 1a: Session termination is performed by sending an HTTP DELETE to the session URL
  - Suboption 1b: Session termination is performed the same way as other session operations
- Option 2: Send POST to session creation URL (potentially including offer), and get HTTP “200 OK” response where the body contains a map from available operations to URLs that the client POSTs to in order to perform those operations (potentially also including answer)

Session Management Properties

- Option 1 in general: Because HTTP defines 201 not to contain a body, we can’t perform offer/answer exchange in first transaction.
    - Suboption 1a: Because there’s not a good mapping from HTTP method to the operations that we know we’ll want, we’ll need to *also* add a generic mechanism for sending WHIP operations. So this approach would make the session termination operation a special case.
    - Suboption 1b: This approach has similar properties as Option 2, with two slight drawbacks: (1) As above, it needs an additional round trip to perform offer/answer exchange, and (2) We would still need to add a mechanism to indicate which operations the server supports.
- Option 2 performs session setup in one round trip, while also indicating supported session operations at the same time.

Session Management Syntax

- I just wanted to add a slide to say that I don’t intend to discuss this topic until we have agreement in principle about which session management approach we want to use, since the overall model is much more important than the syntax we use to implement it.
- However, the BCP 56 prohibition on defining new headers that aren’t applicable at the HTTP layer does mean that some or all of this must go in the HTTP body somehow.

Proposed Extensibility Approach

- WISH document will define a core set of functions that clients and servers must implement.
- These functions must always be sufficient to initiate a broadcast.
- There will be an extension point for adding new operations that can add enhanced functionality.
- There will be an extension point for adding new parameters to existing operations.
- Clients and servers can always ignore unknown or unsupported operations and parameters and still function correctly.

Heartbeats

- Very simply, the client will periodically perform a specified heartbeat operation towards the server, just like other operations.
    - We can specify a hardcoded value in the spec, like 15 seconds
    - Or we can have the server indicate to the client what heartbeat interval it expects

Two-Way Communications

- Minimally, we want the server to be able to terminate a session (and probably other things)
- Approach 1: Send server-to-client operations in heartbeat responses
  - Approach 1a: Just have the server wait for a heartbeat before it can do anything
  - Approach 1b: Have the server hold the heartbeat transaction open (long poll) so that it can respond with an operation whenever it wants
- Approach 2: Set up some persistent channel, like a WebSocket or DataChannel

ICE Management (Trickle & Restarts)

- It is easy to envision future states where communication with IPv4 clients will require even the ingest gateways to be behind NATs, serviced by an application load distributor.
- With a sufficiently rich extension mechanism, we can add this later (although it violates the “base protocol is always sufficient” principle).
- ICE trickle and ICE restart are part of the JSEP state machine, so presumably all WebRTC implementations will include them. Adding them to WHIP adds very little additional complexity.
- While not critical, on the balance, this seems like a good thing to include as base functionality.

Stream Ordinality

- We need clarity around how many streams of each kind a session is expected to contain.
- I think there's an assumption right now that any session will have zero or one video streams, zero or one audio streams, and no application (e.g., datachannel, BFCP, etc) streams.
- As long as we have sufficiently flexible extension points that would allow us to associate two sessions in some way and/or indicate the number of streams a server can handle, this assumption seems sufficient.
- But we definitely want to document it, so I want to make sure we're all on the same page.

WHIP: Additional Feedback

Lorenzo Miniero <lorenzo@meetecho.com>

Lorenzo M.:

"terminate the session": +1
JSON vs HTTP: Mainly neutral, maybe a small preference for JSON
ICE trickling: not opposed.
ICE restarts: +1

“heartbeats for two-way communications”: require details and further discussion

- what kind of commands are expected in the server-to-client direction here?
  Are you mostly thinking about asynchronous events of some nature, or something else too?

[A.R] - Mostly session termination and ICE restarts. But I also want to make sure it's extensible.

I understand the desire to avoid long polls, but

- it may lead to implementations that will poll often and aggressively to be sure they have timely information,
- "forcing" implementations to only send heartbeats at less frequent intervals may lead to suboptimal user experience

Sergio M.:

== Session Termination ==
The need for a way to explicitly terminate the session was already expressed by several people, so I think we should definitely include it.

== Optionalities ==
I would be against optionalities, as it makes the testing and interoperability much more difficult.

== ICE ==
I do not see any practical use case where trickle ICE would be needed today, although I am not against of including it as it would be easy to implement on servers/clients.
I am not convinced about the benefits of an ice restart, compared to just restarting the whole publishing session. The later needs to be implemented in all cases.

== JSON vs HTTP ==
I would prefer to make the process less json-like and more http-like, reusing HTTP headers and methods as much as possible

Others?:

Discussion