JSEP

IETF 92

Changes since last draft

Changes in draft-09:

- o Don't return null for {local,remote}Description after close().
- o Changed TCP/TLS to UDP/DTLS in RTP profile names.
- o Separate out bundle and mux policy.
- o Added specific references to FEC mechanisms.
- o Added canTrickle mechanism.
- o Added section on subsequent answers and, answer options.
- o Added text defining set{Local,Remote}Description behavior

Changes Detail

Issue #114: Policy controls for RTCP-MUX

- HNL: have a separate policy from BUNDLE
- Two policies:
  - `require` - only do RTCP-mux
  - `negotiate` - do mux or non-mux
    - offerer will gather for both RTP and RTCP
- How does answerer `require` behave when offer doesn’t contain mux:
- Resolution: reject track
  - Rationale: consistent behavior for re-offer

SDP: The Good Parts

- JSEP normatively depends on SDP and offer/answer
- (Nearly) any valid SDP should be handled correctly by a JSEP implementation
- Unfortunately, SDP is big and in some cases underspecified
- Objective: provide a map for the territory

Reference the relevant specifications

“First, the session-level lines are checked and parsed. These lines MUST occur in a specific order, and with a specific syntax, as defined in [RFC4566], Section 5” [S 5.6.1]

Restate existing requirements

“Each "m=" and c=" line MUST be filled in with the port and address of the default candidate for the m= section, as described in [RFC5245], Section 4.3” [S 5.3.2]

Fill in gaps

Example: Unified plan

Contradict in rare cases

“Encryption Keys ("k=") lines do not provide sufficient security and MUST NOT be included.”

[S 5.2.1]

“Any profile matching the following patterns MUST be accepted: "RTP/[S]AVP[F]" and "(UDP/TCP)/TLS/RTP/SAVP[F]"” [S 5.2.1]

Processing Lines in Received SDP

1. If the received SDP does not conform to the ABNF, generate an error
2. If the version in the v= line is not correct, generate an error
3. The browser does not have to do anything with the following types of lines:

   s=, i=, u=, e=, p=, t=, r=, z=, c=

4. The browser must generate an error on a k= line
5. The browser will parse and use the information in the o=, a=, and m= lines
6. That leaves just b=. More on b= on the next slide.

Open issue: #27 for auditing whether we need to say anything more about the lines we generate.

The b= SDP

b= is used to signal bandwidth information and can be for whole session or a single media.

b=CT:1234 at a session level would indicate the total proposed bandwidth for all media was 1234 kbps. If this was the only bandwidth signal, the browser could decide how best to use the total bandwidth and split it up between the various tracks.

The rtcweb-rtp-usage draft requires support for b=RS and b=RR (defined in RFC 3556)

The rtcweb-rtp-usage draft requires support for b=AS and b=CT (see section 7.1 of -22)

Proposal:

- AS at media level is the max bitrate the other side wants to receive for that m- line (recommended!)

- AS at session level is not well-defined, and MUST be treated as an error

- CT at session level is the max bitrate for the session, and browser can choose how to deal with this however it sees fit

- CT at media level MUST be treated as an error

#5: use of a=imageattr

Need to be able to convey res/fps limits for a received stream back to the video sender:

- Decoder limits (hard limit)
- App explicitly-set limits (hard or soft limits)
- Video tag size (soft limit)

However, a=imageattr grammar doesn't seem to have this flexibility

#5: use of a=imageattr (2)

Proposal: Add a single a=imageattr:* recv that incorporates

- Decoder max width/height
- Any mandatory width/height constraints

Ignore video tag size, any optional constraints, and any a=imageattr:* send - too complicated.

#5: use of a=imageattr (3)

Example: decoder can decode from 160x90 to 1920x1080; app wants to get between 320x180 and 1280x720. Impl intersects these values to emit this SDP:

```
a=imageattr:* recv [x=[320:1280],y=[180:720]]
```

Issue #72: DTLS role for re-offer

- Want to preserve the roles when you do a re-offer/re-answer
  - **Proposal:** normative MUST strength
- Options
  - Offerer MUST do whatever was negotiated (likely “passive”); Answerer MUST check
  - Offerer MUST do “actpass”; Answerer MUST do the same as before; Offerer MUST check
- **Proposal:** Offerer MUST do negotiated
  - Answerer needs to tolerate actpass for a while

#76: Handling Stopped Tracks

Recap:

- A is sending to B with audio and video
- B stops A's sent video track, while continuing to send its own video
- B generates a new offer: what should m=video contain to indicate the track has been stopped?
  a=recvonly is the only option we have right now, but A doesn't know if this means B paused the track, or stopped it
- Could fix this by adding some new attribute, e.g. a=recvonly-and-never-coming-back, but this is somewhat of an ad-hoc solution

#76: Handling Stopped Tracks (2)

Unclear that B's stopping of the video track matters to A. A will stop sending (due to a=recvonly), and as long as B's impl continues to ignore A's video stream in any reoffer, there should be no issue.

If we want A to get some event in the future about the track being stopped, we can revisit this.

Note that app can also send its own message over signaling.

Proposal: do nothing at this time

Issue #119: How many BUNDLE groups are there?

- Text is sort of unclear
  - Conflict between S 4.1.1 and S 5.2.1

- **Proposal (PR#120):**
  - Always one BUNDLE group for all m= lines
  - `bundle-policy` solely controls which m= lines are marked `bundle-only`
  - Fix 4.1.1 to match this