SDP Offer/Answer Details: Resource & State Management

Adam Roach

Monday, November 11th, Shenzen, China

Sunday, November 10th, Kirkland, WA, USA

What are we talking about?

- When do resources get reserved?
- When do resources get released?
- What operations impact resource state?
- For the most part, I'm hoping we can get additional documentation around these behaviors in the WebRTC API spec.

ICE CANDIDATES

ICE Candidate Gathering

- There are several places where an implementation can reasonably start collecting candidates (number to be collected is impacted by bundling constraint):
  - PC Creation: Optimistically begin gathering candidates for fastest possible connection establishment.
  - AddStream: Could hold off gathering a candidate until we know there's going to be a need for it.
  - CreateOffer / CreateAnswer: Even later in the cycle, allows for being even more conservative by taking into account final configuration of streams and constraints.
  - SetLocalDescription / SetRemoveDescription: Last rational place to gather candidates, since ICE processing needs to start.

Implementation Visibility

- Although the javascript apps could (probably) detect differences among these behaviors if they went looking for them, all of them are consistent with the currently defined API.
- In other words, regardless of what the browser does in this regard, any app written to the spec will behave the same regardless of which of these choices the browser they're running in take.

ICE Candidates: Proposal

- Allow implementations broad discretion in when to begin gathering, so long as key criteria are met:
  1. Candidate gathering begins no later than the success callback for CreateOffer/CreateAnswer.
  2. Candidates for unused media sections are freed upon a successful Set*Description containing an answer.
  3. Unused candidates for used media sections are freed upon ICE completion.

OTHER RESOURCES

Other Resource Reservation (Codecs, Hardware)

- Input resources reserved via gUM
- Input resources released by MediaStreamTrack.stop()
- Codecs reserved by:
  - AddStream
  - CreateOffer with *OfferToReceive*: true*
  - SetRemoteDescription(offer)
- Codecs freed by:
  - Set*Description(answer)
  - RemoveStream
  - MediaStreamTrack.stop()
- Output resources reserved and freed as normal for <audio> and <video> elements

Media Transmission and Reception

- I think this is straightforward and well-understood.
  - Need to be ready to receive as soon as SetLocalDescription is called.
  - Can begin sending as soon as SetRemoteDescription is called.
- This is spelled out in jsep – I don't think it makes sense to reiterate it in the WebRTC spec.

SESSION ROLLBACK

Local Session State Rollback: SetLocalDescription handling

- Call CreateOffer
- Call SetLocalDescription (offer)
- Send offer to remote party
  - At this point, we need to be willing to receive media conforming to either the old description (let's call it "alpha") or the new description (let's call it "beta").
- Receive indication that the remote party rejected our offer
- Need to return local state machine to idle, return session description to pre-offer values
- Jsep-05 suggests calling SetLocalOffer with type="rollback"
  - Presumably, the body of this SetLocalOffer will be "alpha?" You can't ask the PC for this description (is that a bug?). Do we really want to require the javascript to track this?
- Proposal: use type of "rollback," sdp string ignored: SetLocalOffer(new RTCSessionDescription({type:"rollback"}))

Local Session State Rollback: SetRemoteDescription handling

- Jsep-05 also mentions the use of “rollback” in the context of SetRemoteDescription.
  - What use case would necessitate this?
    - SetRemoteDescription(offer): if the JS didn’t want the offer to be processed, it rejects it before calling SetRemoteDescription.
    - SetRemoteDescription(answer): Rolling back is ill-defined. Are we rolling back to the state between the offer and the answer, or the state prior to the offer? How can we possibly expect to get the remote side into a shared state if we do this?
- Proposal: SetRemoteDescription(rollback) always throws an InvalidStateError.

PARTIAL OFFER/PARTIAL ANSWER MECHANISM

Requesting Partial Offers

- Implementations can request partial offers by adding a “partial:true” constraint to createOffer.
- If the changes since the most recent O/A exchange can be expressed as a partial offer, then the RTCSessionDescription passed to the success callback is of type:partialoffer, and the sdp is of the format described in draft-roach-mmusic-pof-pan.
- If the changes require a full O/A exchange, then the success callback is still called, but the RTCSessionDescription is of type:offer.
- If the session is rolled back, it rolls back only the partial offer, not the previous full offer.

Processing Partial Offers

- If setRemoteDescription is called with type:partialoffer, then the subsequent createAnswer *must* generate a type:partialanswer.
  - This does *not* require a partial:true constraint.
  - If additional changes are needed to the session, then an negotiationneeded event is also generated.

CONTROLLING STREAMS IN A SESSION

Stream Pause/Unpause/Rejection/ Removal

- We currently have implementors asking us how to do this. They are more than a little surprised to learn that it's not currently defined in the specification.
- Luckily, I think we can add this functionality without changing the API by simply being clearer about the behavior implied by certain operations
  - And, even if we don't want to add pause/unpause/ reject/remove, we need to be clear about what these operations mean anyway.

Proposal: Pause/Resume *Sending*

- Set enabled=false on a MediaStreamTrack that you have added to a PeerConnection.
  - This information will cause the PeerConnection to stop sending the associated RTP.
  - This will also trigger a negotiationneeded to set the corresponding m-line to recvonly or inactive.
- To unpause, set enabled back to true.
  - The preceding steps are reversed.

Proposal: Pause/Resume *Receiving*

- Set enabled=false on the MediaStreamTrack that you received from the PeerConnection via onaddstream.
  - This will cause the MST to stop providing media to whatever sink it has been wired to.
  - This also triggers a negotiationneeded event. The subsequent CreateOffer sets the corresponding m-line to sendonly or inactive, as appropriate.
- Unpause by setting enabled back to true.

Proposal: Rejecting an Offered Stream

- To reject a track that has been offered, call stop() on the corresponding MediaStreamTrack *after* it has been received via onaddstream, but *before* calling CreateAnswer. This will cause it to be rejected with a port number of zero.

Proposal: Removing a Stream

- Call stop() on the corresponding MediaStreamTrack object.
  - This will immediately stop transmitting associated RTP.
  - This will also trigger a negotiationneeded event. If both the sending and receiving MST associated with that m-line have been stop()ed, then the subsequent CreateOffer sets the port on the corresponding m-line to zero.
- If only one of the two MSTs associated with that m-line has been stop()ed, then the subsequent CreateOffer sets the corresponding m-line to sendonly, receiveonly, or inactive, as appropriate.

ONE MORE TINY ISSUE

Minor Issue: SDP Operation Queuing Spec Bug

- Current spec: “The general idea is to have only one among createOffer, setLocalDescription, createAnswer and setRemoteDescription executing at any given time. If subsequent calls are made while one of them is still executing, they are added to a queue and processed when the previous operation is fully completed.”
- I’m pretty sure this list is missing addIceCandidate.
  - Consider queuing setRemoteDescription/ setRemoteDescription/addIceCandidate: which sRD should the aIC apply to?
- Proposal: add addIceCandidate to the list.