RTCWeb Signaling, ROAP, and the PeerConnection API

IETF 82

Cullen Jennings

The starting point for ROAP

- Old W3C API had PeerConnection that emits messages like this:

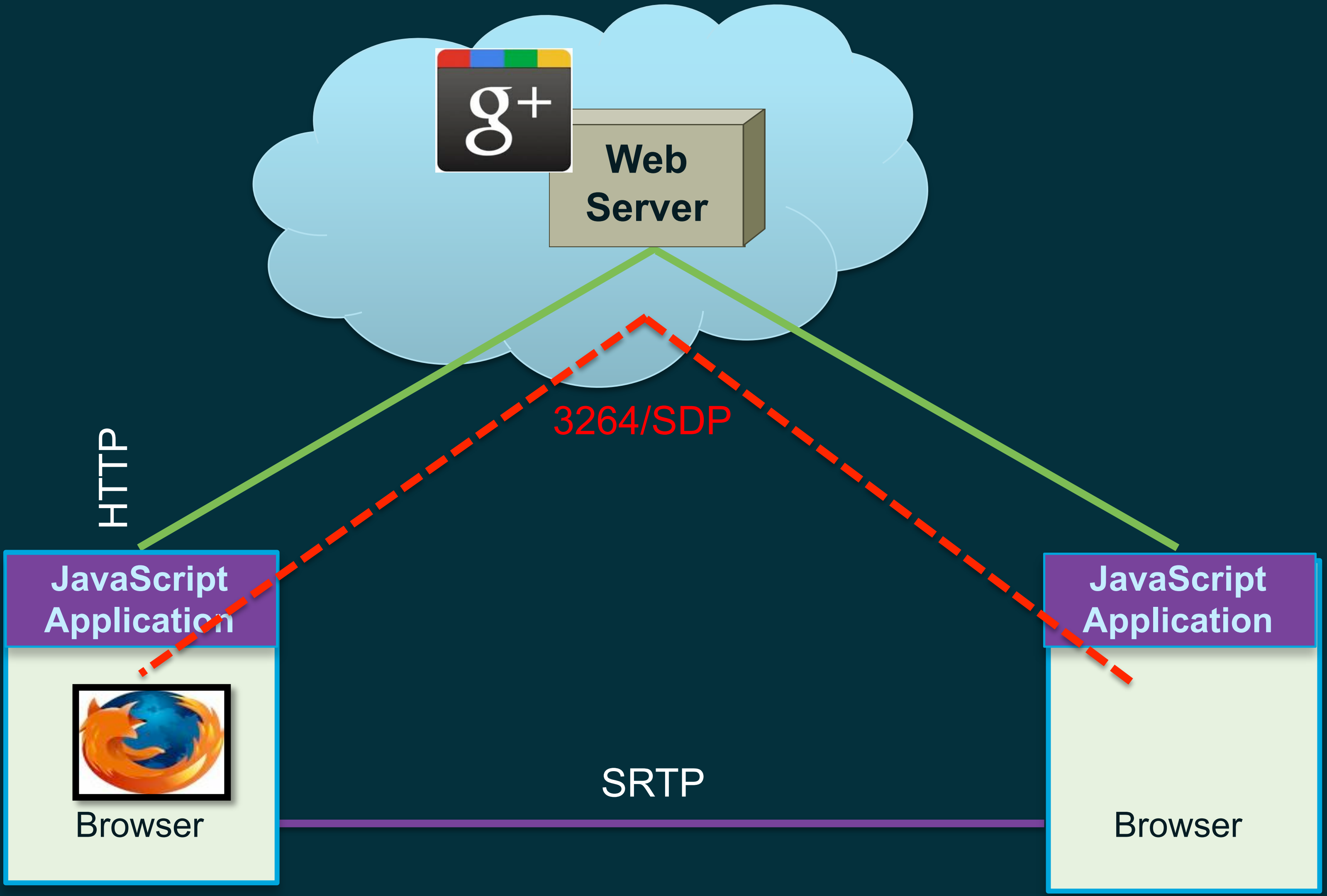
```
“SDP\n
  v=0\n
   o=- 2890844526 2890842807 IN IP4 192.0.2.1\n
   s= \n
   c=IN IP4 192.0.2.1\n
   t=2873397496 2873404696\n
   m=audio 49170 RTP/AVP 0”
```

ROAP adds minimal fields to make RFC 3264 work.

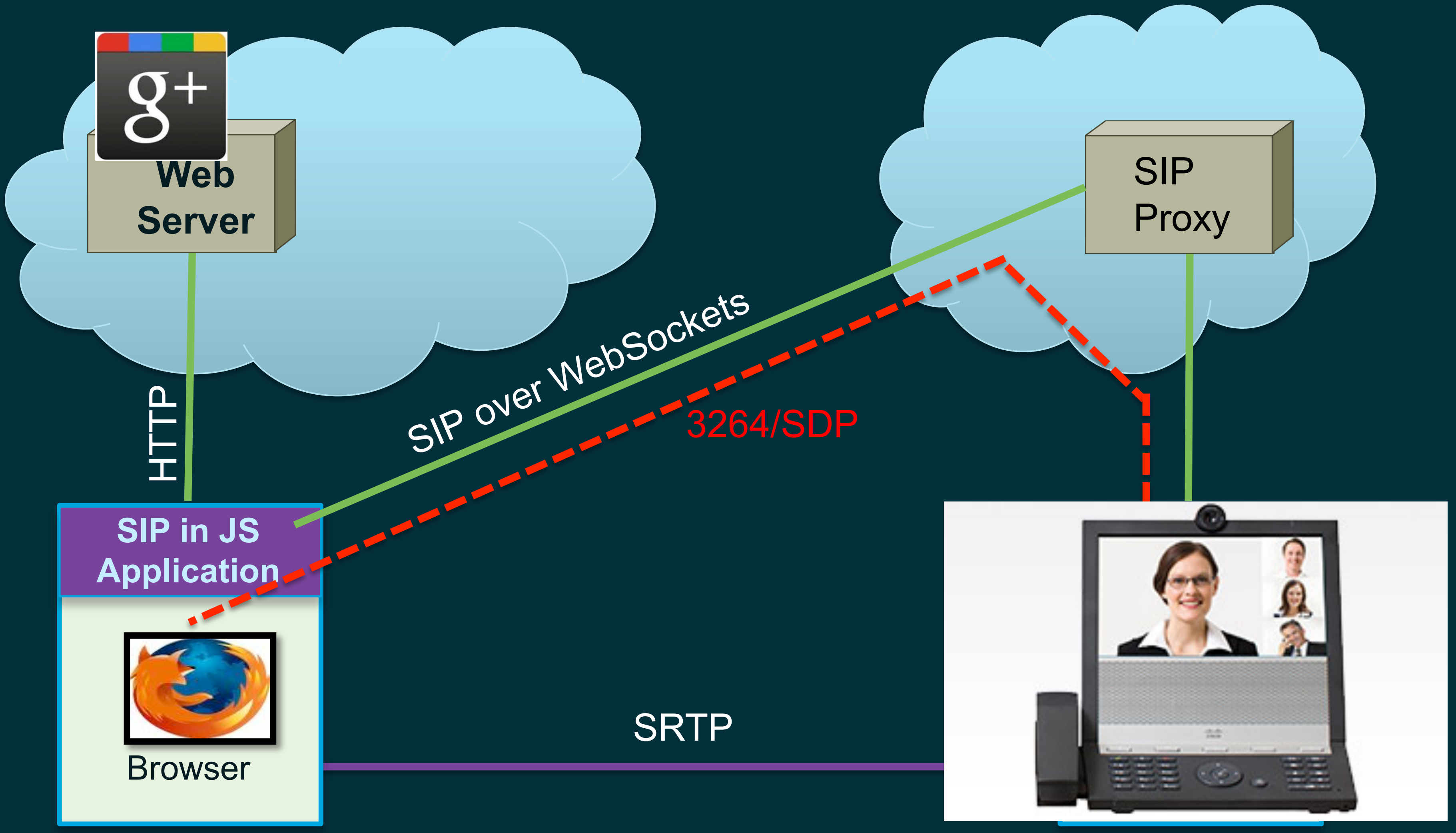
- More on each of these fields later ….
- {

```
  “messageType”: “OFFER”,
  "offererSessionId":"13456789ABCDEF",
  “seq”: 1,
  “sdp”:”v=0\n
    o=- 2890844526 2890842807 IN IP4 192.0.2.1\n
    s= \n
    c=IN IP4 192.0.2.1\n
    t=2873397496 2873404696\n
    m=audio 49170 RTP/AVP 0”
  }
```

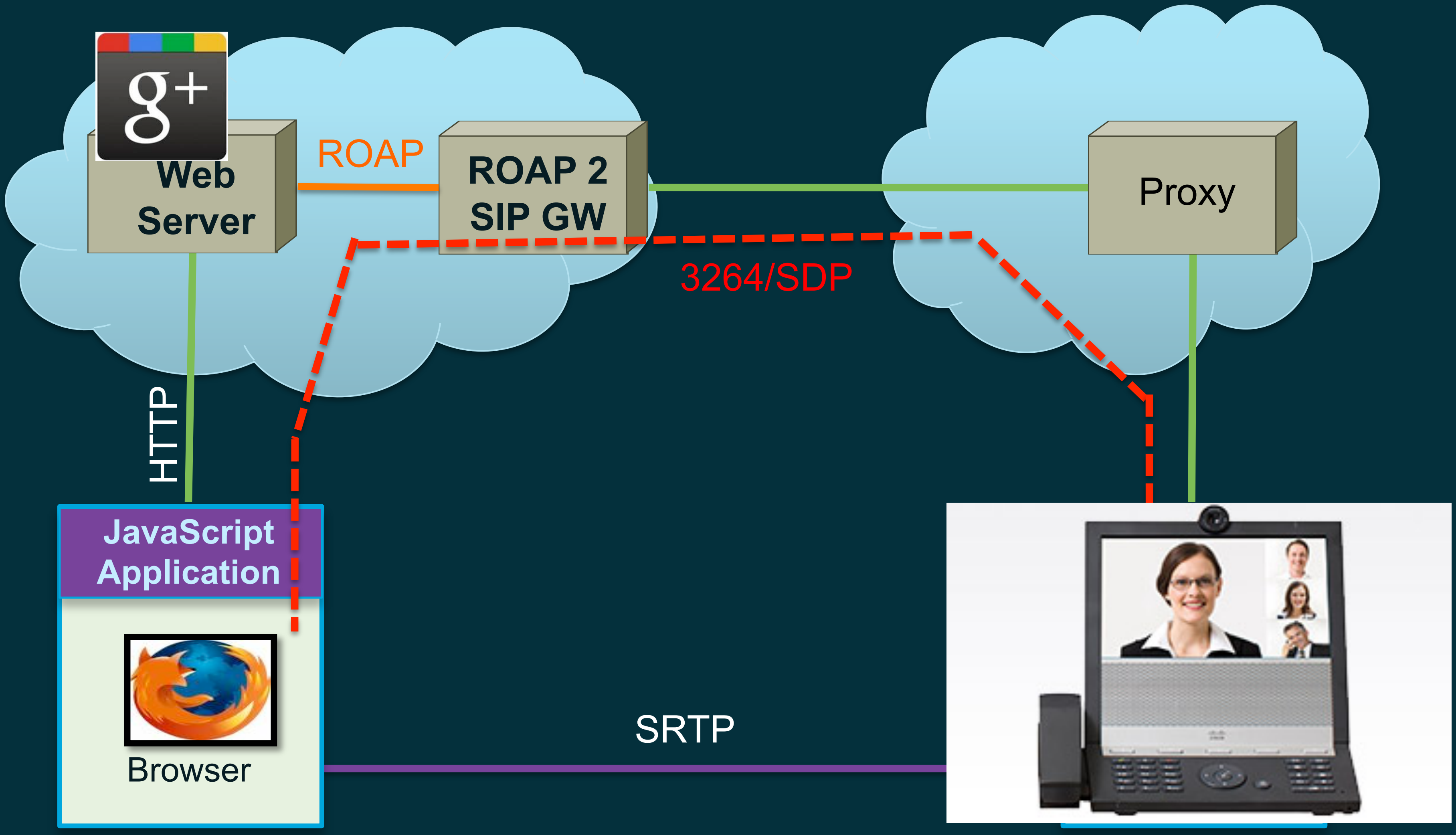
Architecture



This Architecture is also OK …



Architecture – Media Negotiation to Legacy



Adding fields to basic SDP … A JSON version

- {
```
  “sdp”:”v=0\n
    o=- 2890844526 2890842807 IN IP4 192.0.2.1\n
    s= \n
    c=IN IP4 192.0.2.1\n
    t=2873397496 2873404696\n
    m=audio 49170 RTP/AVP 0”
  }
```
- Now at least we can extend things

How do we tell offers from answers?

- Easy to get confused if we have glare
- We both send SDP at the same time
- Is the SDP I just got a new call or an answer to my offer?

Message types

- {

  “messageType”: “OFFER”,

  “sdp”:”v=0\n

    o=- 2890844526 2890842807 IN IP4 192.0.2.1\n

    s= \n

    c=IN IP4 192.0.2.1\n

    t=2873397496 2873404696\n

    m=audio 49170 RTP/AVP 0”

  }

How do we identify sessions?

(By session I just mean some vague fluffy hand wavy sort of concept of a set of audio and video data flowing to some other user or application – I don’t mean what RTP or DTLS might mean)

- We might have multiple sessions going on
- Either simultaneously or in sequence
- Message delays can make these look like each other

Session ID

- {
```
   “messageType”: “ANSWER”,
    "offererSessionId":"13456789ABCDEF",
    "answererSessionId":"abc1234356",
     “sdp”:”v=0\n
     o=- 2890844526 2890842807 IN IP4 192.0.2.1\n
     s= \n
     c=IN IP4 192.0.2.1\n
     t=2873397496 2873404696\n
     m=audio 49170 RTP/AVP 0”
  }
```
- Session ID must be globally unique

How session IDs get established

- {
  “messageType”: “OFFER”,
  "offererSessionId":"13456789ABCDEF”, …
  }
- {
  “messageType”: “ANSWER”,
  "offererSessionId":"13456789ABCDEF",
  "answererSessionId":"abc1234356”, …
  }
- Each side contributes a session ID
- Session defined by combination of each side – unique to current offer/answer pair on given session
  - Forked calls would result in two session

Multiple offer/answer pairs

- A session has a sequence of offer/answer pairs
- Example: upgrade to video
  - We have an audio call
  - The users decide to add video
- This requires updating the offer/answer pair
- How do we distinguish them?

Sequence field

- {
  “messageType”: “OFFER”,
  "offererSessionId":"13456789ABCDEF”,
  “seq”: 1,
  “sdp”:”…”
  }
- Sequence indicates the current offer/answer exchange
  - “Generation” may be better name for this than “sequence”
- OFFER and ANSWER have same sequence number

ANSWER Confirmation

- It’s not safe to have multiple OFFERs outstanding
- What happens if I do two changes in succession?
- We can get glare (more on that later) but need way to know there was not glare
- Example:
  - Other side adds video
  - I accept but then user changes camera to one with different capabilities
  - I need to re-OFFER but when?
  - After he’s gotten my ANSWER

OK Message

- {
  “messageType”: “OK”,
  “seq”:”2”,
  "offererSessionId":"13456789ABCDEF”,
  "answererSessionId":"abc1234356”
  }
- OK message indicates ANSWER received and accepted
- New session parameters are active
- Safe to do a new OFFER/ANSWER pair

ICE Pipelining

- ICE is slow
- We want to start ICE as soon as possible
- Best experience is to start ICE when the browser receives the OFFER even before the user accepts the call
- But we don't know the media parameters till the user answers the call
  - For instance, the user might accept audio but not video
- (Something like the proposed mechanism is also needed to gateway to 1-800-gofedex use case)

moreComing flag

- {

  “messageType”: “ANSWER”,

  "offererSessionId":"13456789ABCDEF",

  "answererSessionId":"abc1234356",

  “moreComing”: true,

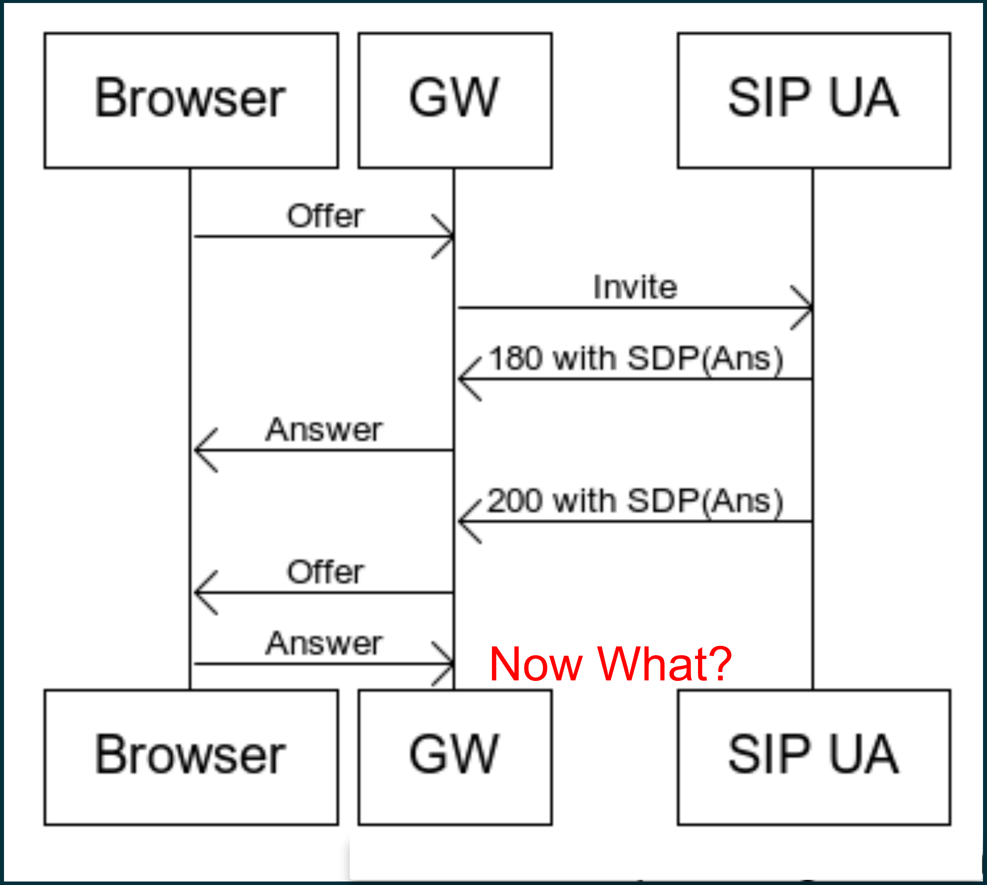
  “sdp”:…   // ICE candidates but recvonly media

  }
- moreComing flag means that another ANSWER will follow to this OFFER
- OFFER/ANSWER transaction isn’t complete till moreComing=false
- No OK for moreComing=true

1-800-gofex

- Also can use moreComing flag to gateway to early media
- High level had 3 complicated cases
  - A non final offer such as moreComing
  - Serial forking
  - Parallel forking
- Forking can result in new sessionIDs
- moreComing keeps the offer from changing while waiting for an updated Answer on same sessionID or other Answers with new sessionID

Alternative design with media GW



Glare

- See slides in next presentation …
  - Adds a few more error type to support glare negotiation

State Cleanup

- Headless browsers pages, buggy JS, and general JS garbage collection make state management a non trivial issue
- Two situations:
  - one where the far side just fell off the network
  - Second where an orderly shutdown is in process and one side is telling other side to clean up the state in the SDP Agent
- Need to to display different errors to user
- Order shutdown provides more reliable and timely state clean up
- Timely clean up can be important to free up a resources like a camera for use by another session

SHUTDOWN Message

- {

  “messageType”: “SHUTDOWN”,

  “seq”:”2”,

  "offererSessionId":"13456789ABCDEF”,

  "answererSessionId":"abc1234356”

  }
- Allows both sides to understand the media is being deliberately removed, clean up state, and not display the types of errors to users that might be displayed if the RTP connectivity was lost
- Need to design when things “go away” in clean exit. For example:

  In a “clean” exit, both sides wait until they get a OK to the shutdown message they sent. When B receives a SHUTDOWN, it sends a SHUTDOWN to A then B waits for OK from A. When B receives the OK, it then sends the OK for original SHUTDOWN from A.

Indicating Capabilities

- This is not in ROAP but we could add it …
- RFC 3264 supports a concept of Indicating Capabilities
- SIP uses this in the SIP OPTION message
- This allows a remote way to find out about browser support for various codecs
- Privacy: This is a issue for browser fingerprinting. This function, if implemented by browsers, would most likely worsen the privacy situation

Capabilities Message

- {
  “messageType”: “CAPS-REQ”,
  … (still vague on how session id works here)
  }
- {
  “messageType”: “CAPS”, … ,
  “sdp”:”v=0\n
  o=- 2890844526 2890842807 IN IP4 192.0.2.1\n
  s= \n
  c=IN IP4 192.0.2.1\n
  t=0 0\n
  m=audio 0 RTP/AVP 0”
  }

Hints

- Tell CODECs something about the application that they may need to understand to make good encoding choices
- Audio: is music or is spoken voice
- Video: prefer spatial or temporal fidelity
- Proposal
  - IANA registry of well known hints
  - Have some sort of setHints method on a media stream
- Can apply before session starts or during session

Stats

- Proposal
- Have a stats method on PeerConnection
- Have it return a dictionary of stats
- Define an IANA registry of well known stat names
  - Total RTP packets received
  - Total missing RTP packets
  - …
- The value for a stat can be primitive type, array or object

ROAP to SIP Signaling Gateway

- draft-jennings-rtcweb-signaling-gateway has sketch of design
- Goal is to allow a stateless GW design by passing the GW state to the browser to be returned to the GW later.
- State data is opaque blob to browser