A Unified Plan for SDP Handling

Adam Roach

Berlin, Germany

Wednesday, July 31, 2013

Motherhood and Apple Pie

- Large number of arbitrary sources
- Fine-grained receiver control of sources
- Glareless addition and removal of sources
- Interworking with non-WebRTC devices
- Avoidance of excessive port allocation
- Simple binding of MediaStreamTrack to SDP
- RTX, FEC, simulcast, layered coding

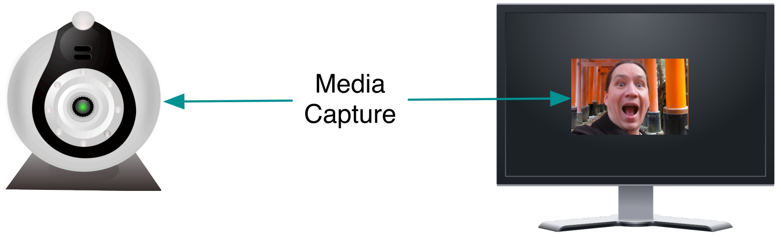
Planks in the Plan

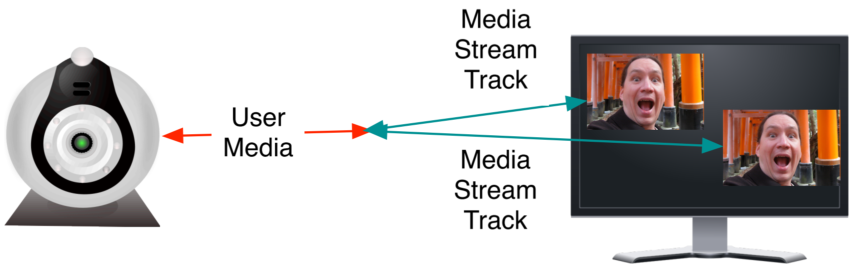
1. One m-line == one MediaStreamTrack
2. Each m-line has (at least one) a=ssrc for correlation*
3. Each m-line has an MSID to correlate it with a MediaStream and MediaStreamTrack
4. Mid-call port allocation is minimized using BUNDLE
5. Call startup port allocation is minimized through “BUNDLE-only” lines
6. Glare is addressed via a “Partial Offer/Answer” extension
7. Transport-wide attributes are identical for every m-line in the same bundle
8. Additional mechanisms for early identification of incoming media streams are defined (new RTP extension header; and, exceptionally, the use of unique PTs)

*When possible. Some architectures may preclude doing so. Those implementations will possibly miss out on important features.

MediaStreamTrack? I can't read your crazy moon language!

- Defined in W3C docs
- Kind of the same thing as a CLUE "Media capture"
- Except that Web applications can, e.g., take the output of a camera, split it, and make more than one MediaStreamTrack out of it.





Let’s Point Out The Elephant

- There’s a lot of SDP out there in the real world.
- Lots of people have done varying things with SDP -- some supported by RFCs, others not -- that comprise a wide and conflicting corpus of “existing SDP usage.”
- In many cases, when trying to nail down historically ambiguous usage, we’ve had to choose a behavior that fits with some existing uses, and which conflicts with others.
- Largely, we’ve tried to go for the solutions that make the most sense with the rest of our proposal, are most supported by existing RFCs, or are most widely deployed.
- No matter which direction these decisions take, *someone* will want to stand up and say “but that’s not how mine works.” We know. We’re sorry.
  - We’re happy to talk about different answers to these questions, but we can’t expect to start with systems that interpret the same SDP as meaning radically different things and walk away with a fully backwards-compatible, consistent system. Someone will be unhappy.

M-line == MediaStreamTrack

- We’ve taken the smallest thing that we typically need to control and made it the unit that we deal with in SDP.
- This is congruent with many (although admittedly not all) deployed SDP offer/answer uses.

Use of a=ssrc

- Every m-line has (at least one) a=ssrc in it, to specifically tie it to its corresponding RTP packets.
  - Although it is possible to get SDP from non-unified-plan clients that omits it, and we should react in a sane fashion.
- An implication of “one MediaStreamTrack per m-line” is that any SSRC associated with an m-line beyond the first one needs to be explained some way.
  - That is: since this is only one “thing,” why are you sending more than one stream for it?
- To assist with a commonly deployed (although not universal) usage of multiple SSRCs, the default interpretation when you get two SSRCs on an m-line is to treat them as “tag-teaming” each other.
- *Yeah, there’s a bug in the examples: we need to add a=ssrc:cname. We’ll fix that.*

Media Correlation

- Uses a=msid to tie application-level thing to m-line

  Yes, this is a change to the msid draft. It makes sense once you do one-m-line-per-MST

- For WebRTC, this will be in the format <MediaStreamId> <space> <MediaStreamTrackId>

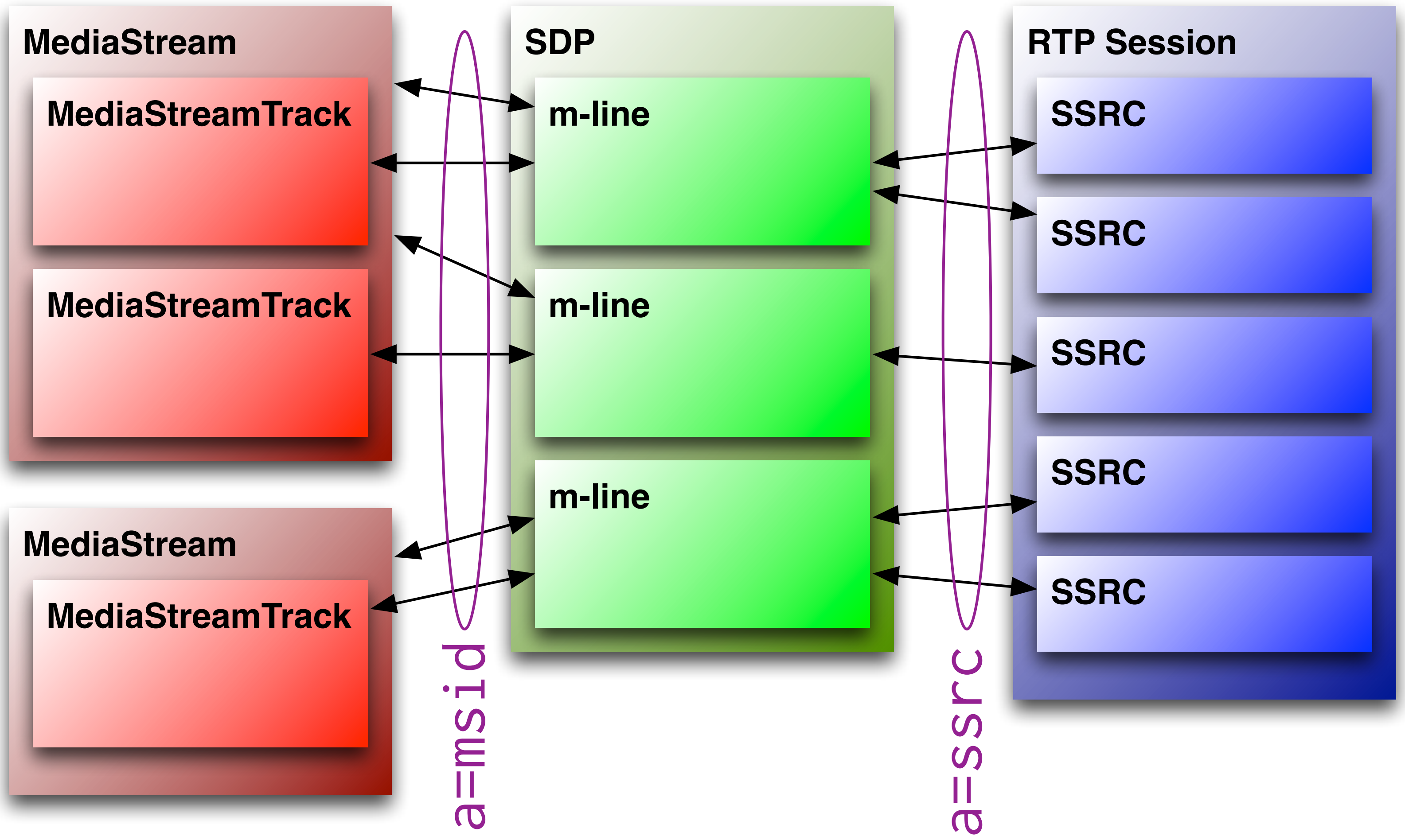
Example: Two media streams (ma and mb). Stream ma has two tracks (ta and tb), while stream mb has one track (tc):

```
m=audio 54400 RTP/SAVPF 0 96
a=msid:ma ta

m=video 0 RTP/SAVPF 96 97
a=msid:ma tb

m=video 0 RTP/SAVPF 96 97
a=msid:mb tc
```

Correlation of what? I got lost.



Port Use Reduction

- We normatively depend on BUNDLE draft-ietf-mmusic-sdp-bundle-negotiation for reduced port use mid-call.
- We use the Plan A “Bundle-Only” lines for m-lines that are allowed to fail when talking to non-bundling clients.
  – *Yes, we know that this requires changes to normative statements the BUNDLE draft (e.g., allowing port=0 in an offer). BUNDLE is still a draft; MMUSIC has the power to change it.*

Bundle-Only Example

So, if you wanted to set up a call that had one audio stream and two video streams, but only wanted video if you're talking to a bundle client…

```
a=group:BUNDLE S1 S2 S3
…
m=audio 54400 RTP/SAVPF 0 96
a=mid:S1
…
m=video 0 RTP/SAVPF 96 97
a=mid:S2
a=bundle-only
…
m=video 0 RTP/SAVPF 96 97
a=mid:S3
a=bundle-only
```

Open Issue: Default Behavior for "Bundle Only"

- The indication of which streams are "Bundle Only" is intended to be supported through the use of constraints (at least, in WebRTC).
- We do not define which behavior is used if no constraint is present; I think there are three supportable positions:
  1. The first audio stream is independent, all others are bundle-only (except for video-only calls, in which the first video stream is independent)
  2. The first stream of each media type is independent, all others are bundle-only
  3. All streams are independent unless explicitly constrained
- This may well be a W3C issue anyway.

Glare Reduction

- We need to be able to add and remove streams without glare conditions arising.
  - It would be <u>nice</u> if we could make it unlikely for stream modification, too.
- The draft contains a *thumbnail sketch* of an approach that allows glareless addition and removal of streams.
- The approach isn't key to the plan, but having *an* approach is.
- If we don't like this approach, we can design a different one (or use one of the two others that have been discussed on-list).

Glare Reduction: Partial Offer/ Partial Answer

- For an ongoing call, applications can request a “partial offer” that contains just the portions of SDP that have changed.
- This partial offer can be correlated to a stream (or determined to be a new stream) using its MID.
- New streams are appended to the SDP
  - If two streams are added, we have a tiebreaker
- Removed streams are set inactive
  - If the stream is removed and changed at the same time, the removal “wins”
- Changed streams are changed
  - If both sides try to change the same stream simultaneously, then glare resolution is necessary.
  - This is an improvement over normal 3264, where both sides trying to change the *session* simultaneously causes glare.
- The response to a “partial offer” is a “partial answer”: it contains exactly the same m-line(s) as the partial offer (no more, no less).

SDP Attribute Handling

- All m-lines in a bundle contain the same attributes except for those which apply directly to streams.
- The unified plan proposes draft-nandakumar-mmusic-sdp-mux-attributes as the basis for characterizing which attributes fall under this umbrella.

RTX, FEC, Simulcast, and Layered Coding

- These techniques produce multiple RTP streams for a single MediaStreamTrack.
- Remember, we're planning on one m-line per media stream track.
- RTX, FEC, and layered coding are already described in RFC 4588, RFC 5956, and RFC 5583 respectively.
  - We use exactly the syntax from RFC 4588 §8.8 and RFC 5956 §4.3.
  - For RTX, we add a=ssrc-group:FID to make pairings explicit (needed for RTX & simulcast at the same time).
  - We use the syntax from RFC 5583 (albeit with all streams in the same m-line)
- SDP for simulcast is not defined in an RFC or an adopted draft yet. We propose the use of a=ssrc-group:SIMULCAST to tie these together.

Simulcast Example

```
m=video 62537 RTP/SAVPF 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 15955
a=mid:1
a=rtpmap:96 VP8/90000
a=sendrecv
a=rtcp-mux
a=ssrc:29154375 imageattr:96 [x=1280,y=720]
a=ssrc:47182014 imageattr:96 [x=640,y=360]
a=ssrc-group:SIMULCAST 29154375 47182014
```

Simulcast Handling

- Note that the PTs are the same for both streams.
  - Unless they need different fmtp parameters for some reason.
- If you are sending inband parameter sets (or using VP8), you can tell the streams apart by looking at the resolution in the initial IDR.
- To reject one simulcast stream, one approach could be using imageattr to indicate only a single recv resolution is desired.
- Regardless of how it's signaled, the offerer can really only reject simulcast in a second O/A exchange.

Simulcast Example w/FEC

```
m=video 62537 RTP/SAVPF 96 101
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 15955
a=mid:1
a=rtpmap:96 VP8/90000
a=rtpmap:101 1d-interleaved-parityfec/90000
a=sendrecv
a=rtcp-mux
a=ssrc:29154375 ...
a=ssrc:47182014 ...
a=ssrc:38259631...
a=ssrc:18697302 ...
a=ssrc-group:SIMULCAST 29154375 47182014
a=ssrc-group:FEC-FR 29154375 38259631
a=ssrc-group:FEC-FR 47182014 18697302
```

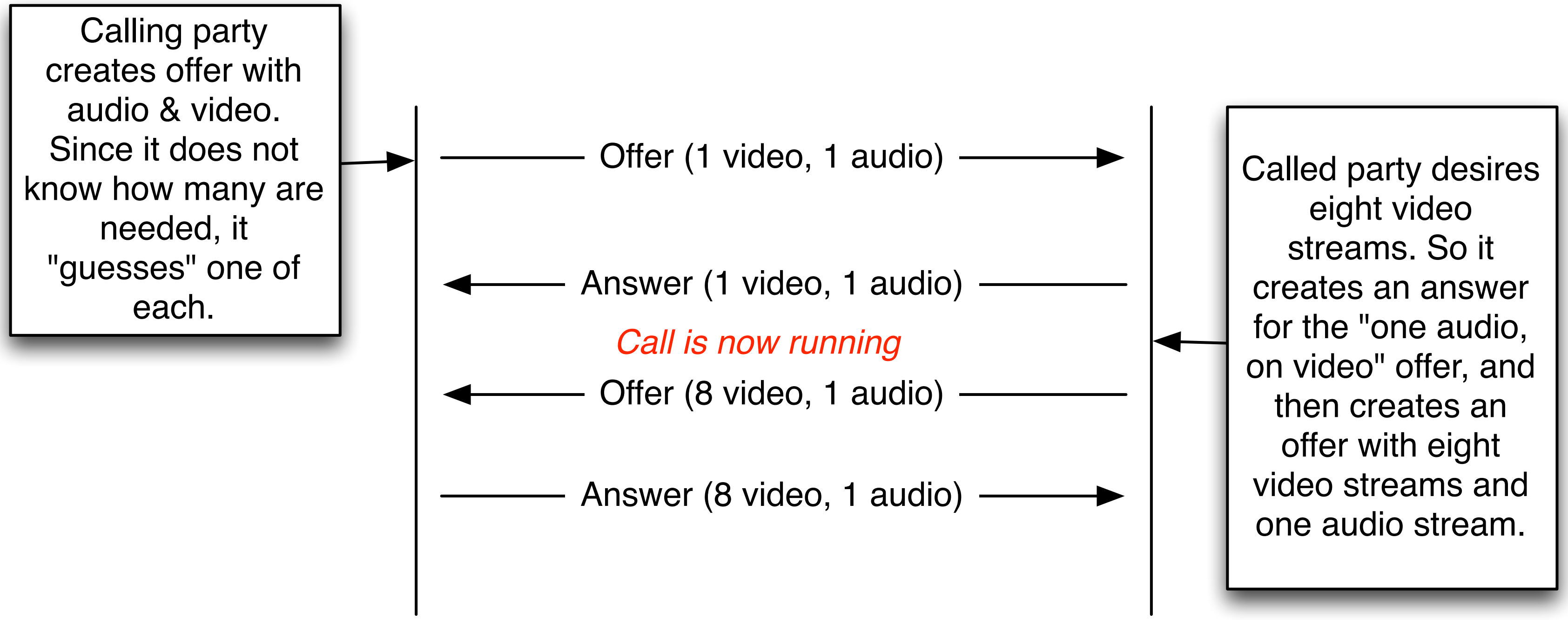
Simulcast Example w/RTX

```
m=video 62537 RTP/SAVPF 96 101
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 15955
a=mid:1
a=rtpmap:96 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=96;rtx-time=3000
a=sendrecv
a=rtcp-mux
a=ssrc:29154375 ...
a=ssrc:47182014 ...
a=ssrc:38259631...
a=ssrc:18697302 ...
a=ssrc-group:SIMULCAST 29154375 47182014
a=ssrc-group:FID 29154375 38259631
a=ssrc-group:FID 47182014 18697302
```

“Handshaking”

1. Calling party sends an offer with at least one m-line for each media type it wants in the call (more, if it thinks it might help).
2. Called party uses as many streams as are present. If there are enough for its needs, then no further action is needed.
   - *The call is set up at this point*
3. If the called party needs more streams, it sends an offer (probably partial) in the other direction immediately, increasing the stream count as needed.
4. The calling party processes this (partial) offer normally, and sends an appropriate (partial) answer.

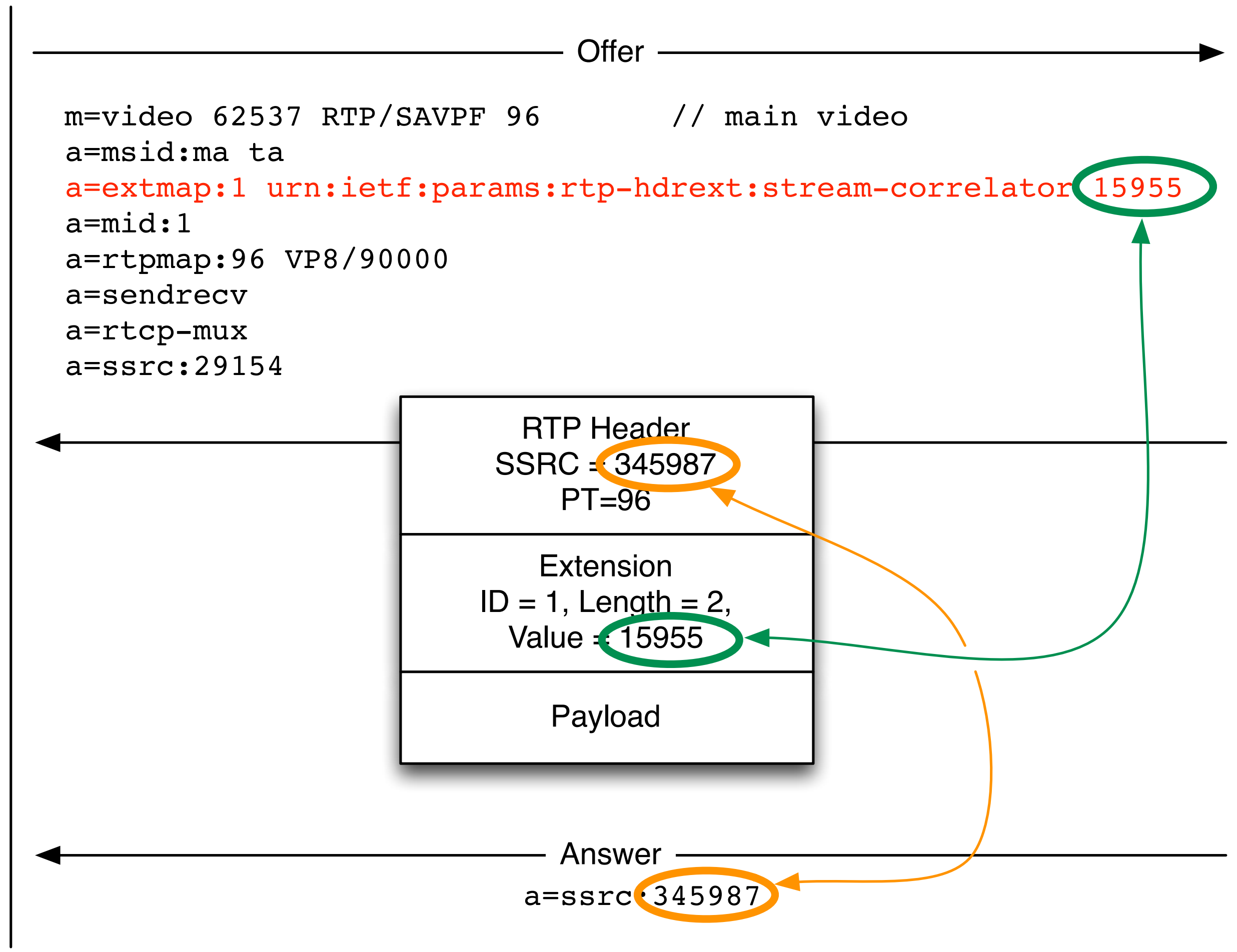
“Handshaking”


Calling party creates offer with audio & video. Since it does not know how many are needed, it "guesses" one of each.
Offer (1 video, 1 audio)
Answer (1 video, 1 audio)
Call is now running
Offer (8 video, 1 audio)
Answer (8 video, 1 audio)
Called party desires eight video streams. So it creates an answer for the "one audio, on video" offer, and then creates an offer with eight video streams and one audio stream.

Matching RTP Streams to m-lines

- Primarily, done through a=ssrc in each m-line
- Secondarily, to handle certain races, performed via an RFC 5285 RTP header extension field. The value is chosen by the party who will be *receiving* the corresponding media.
- Exceptionally, performed by using unique payload types.

Matching RTP Streams to m-lines: RTP Header Extension



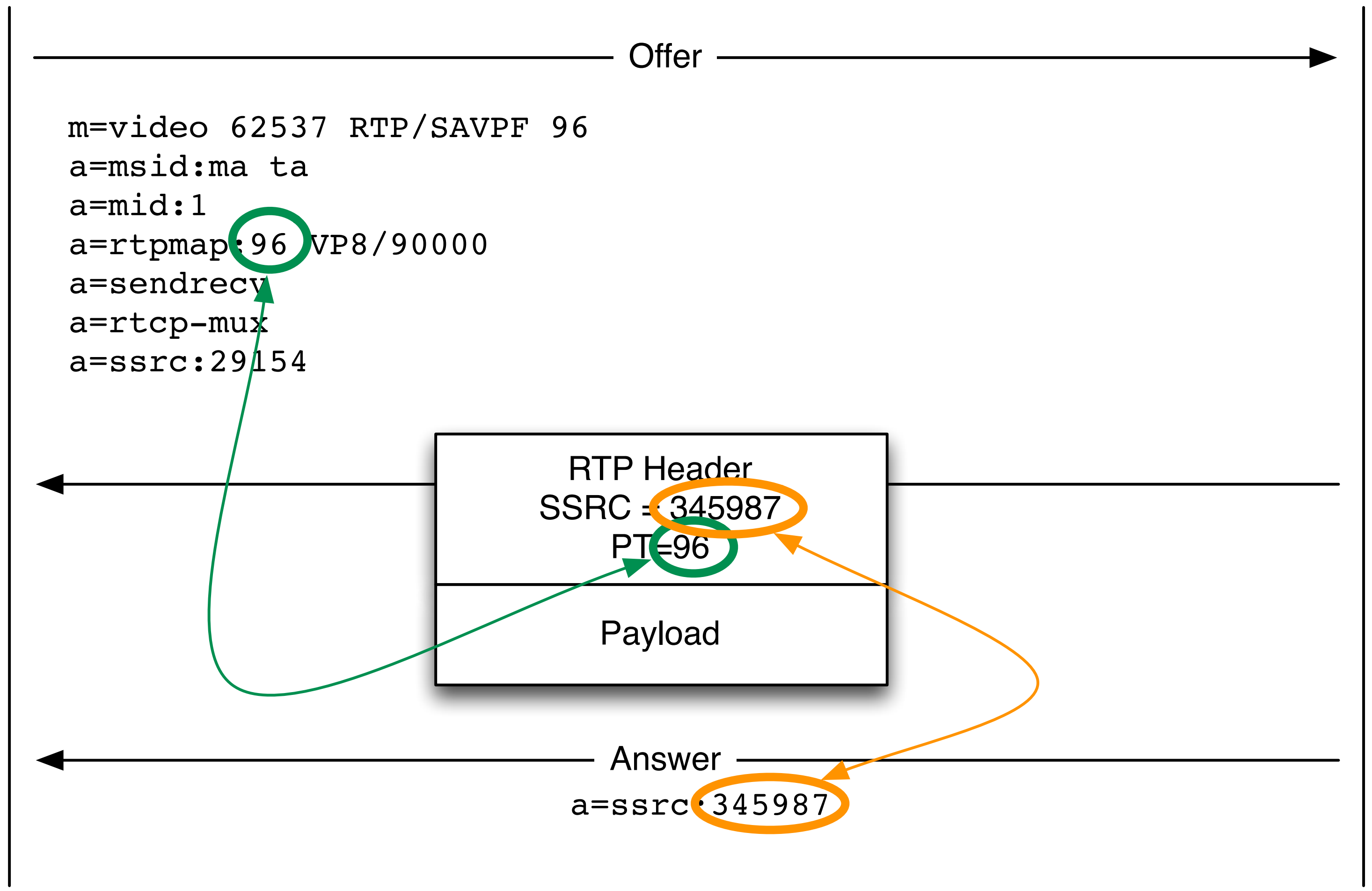
```
m=video 62537 RTP/SAVPF 96        // main video
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 15955
a=mid:1
a=rtpmap:96 VP8/90000
a=sendrecv
a=rtcp-mux
a=ssrc:29154
```

```
a=ssrc:345987
```

Matching RTP Streams to m-lines: Unique Payload Types

- Less-preferred, included for systems that cannot (easily) use RTP extension
- In WebRTC, applications would need to add a constraint explicitly requesting this behavior
- When constraint is present, browser tries to make PTs completely unique so they can be used to correlate m-lines
  - If it can't, it reports an error to the application
- When the constraint is absent, PTs are re-used between m-lines.

Matching RTP Streams to m-lines: Unique Payload Types



```
m=video 62537 RTP/SAVPF 96
a=msid:ma ta
a=mid:1
a=rtpmap:96 VP8/90000
a=sendrecv
a=rtcp-mux
a=ssrc:29154
```

```
a=ssrc:345987
```

Open Issue: What PTs are Okay?

PT	Encoding Name	A/V
0	PCMU	A
1	Reserved	
2	Reserved	
3	GSM	A
4	G723	A
5	DVI4	A
6	DVI4	A
7	LPC	A
8	PCMA	A
9	G722	A
10	L16	A
11	L16	A
12	QCELP	A
13	CN	A
14	MPA	A
15	G728	A
16	DVI4	A
17	DVI4	A
18	G729	A
19	Reserved	A
20	Unassigned	A

PT	Encoding Name	A/V
21	Unassigned	A
22	Unassigned	A
23	Unassigned	A
24	Unassigned	V
25	CelB	V
26	JPEG	V
27	Unassigned	V
28	nv	V
29	Unassigned	V
30	Unassigned	V
31	H261	V
32	MPV	V
33	MP2T	AV
34	H263	V
35-63	Unassigned	?
64-95	Reserved for RTCP conflict avoidance	
96-127	dynamic	?