Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 16, 2014

A. B. Roach
Mozilla
J. Uberti
Google
M. Thomson
Microsoft

July 15, 2013

A Unified Plan for Using SDP with Large Numbers of Media Flows
draft-roach-mmusic-unified-plan-00

Abstract

A recurrent theme in emerging real-time communications use cases, such as RTCWEB, has been the need to handle very large numbers of media flows. Unfortunately, naive uses of SDP do not handle this case particularly well. This document describes a modest set of extensions to SDP which allow it to cleanly handle arbitrary numbers of flows while still retaining a large degree of backward compatibility with existing and non-RTCWEB endpoints.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 16, 2014.

Table of Contents

1. Introduction

A recurrent theme in new RTC technologies has been the need to cleanly handle very large numbers of media flows. For instance, a videoconferencing application might have a main display plus thumbnails for 10 or more other speakers all displayed at the same time. If each video source is encoded in multiple resolutions (e.g., simulcast or layered coding) and also has FEC or RTX, this could easily add up to 30 or more independent RTP flows.

This document focuses on the WebRTC use cases, and uses its terminology to discuss key concepts. The approach described herein, however, is not intended to be WebRTC specific, and should be generalize to other SDP-using applications.

The standard way of encoding this information in SDP is to have each RTP flow (i.e., SSRC) appear on its own m-line. For instance, the SDP for two cameras with audio from a device with a public IP address could look something like:

```
v=0
o=- 20518 0 IN IP4 203.0.113.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
        42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7

m=audio 54400 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 52595
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=candidate:0 1 UDP 2113667327 203.0.113.1 54400 typ host
a=candidate:1 2 UDP 2113667326 203.0.113.1 54401 typ host

m=video 55400 RTP/SAVPF 96 97
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 56036
a=rtpmap:96 H264/90000
```

```
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
a=candidate:0 1 UDP 2113667327 203.0.113.1 55400 typ host
a=candidate:1 2 UDP 2113667326 203.0.113.1 55401 typ host

m=video 56400 RTP/SAVPF 96 97
a=msid:ma tc
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 21909
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
a=candidate:0 1 UDP 2113667327 203.0.113.1 56400 typ host
a=candidate:1 2 UDP 2113667326 203.0.113.1 56401 typ host
```

Unfortunately, as the number of independent media sources starts to increase, the scaling properties of this approach become problematic. In particular, SDP currently requires that each m-line have its own transport parameters (port, ICE candidates, etc.), which can get expensive. For instance, the [RFC5245] pacing algorithm requires that new STUN transactions be started no more frequently than 20 ms; with 30 RTP flows, which would add 600 ms of latency for candidate gathering alone. Moreover, having 30 persistent flows might lead to excessive consumption of NAT binding resources.

This document specifies a small number of modest extensions to SDP which are intended to reduce the transport impact of using a large number of flows. The general design philosophy is to maintain the existing SDP negotiation model (inventing as few new mechanisms as possible) while simply reducing the consumption of network resources.

1.1. Design Goals

The mechanism described in this document is meant to address the following goals:

1.1.1. Support for a large number of arbitrary sources

In cases such as a video conference, there may be dozens or hundreds of participants, each with their own audio and video sources. A participant may even want to browse conferences before joining one, meaning that there may be cases where there are many such conferences displayed simultaneously.

In these conferences, participants may have varying capabilities and therefore video resolutions. In addition, depending on conference

policy, user preference, and the desired UI, participants may be displayed in various layouts, including:

- A single large main speaker with thumbnails for other participants
- Multiple medium-sized main speakers, with or without thumbnails
- Large slides + medium speaker, without thumbnails

These layouts can change dynamically, depending on the conference content and the preferences of the receiver. As such, there are not well-defined 'roles', that could be used to group sources into specific 'large' or 'thumbnail' categories. As such, the requirement we attempt to satisfy is support for sending and receiving up to hundreds of simultaneous, heterogeneous sources.

1.1.2. Support for fine-grained receiver control of sources

Since there may be large numbers of sources, which can be displayed in different layouts, it is imperative that the receiver can easily control which sources are received, and what resolution or quality is desired for each (for both audio and video). The receiver should also be able to prioritize the source it requests, so that if system limits or bandwidth force a reduction in quality, the sources chosen by the receiver as important will receive the best quality. These details must be exposed to the application via the API.

1.1.3. Glareless addition and removal of sources

Sources may come and go frequently, as is the case in a conference where various participants are presenting, or an interaction between multiple distributed conference servers. Because of this, it is desirable that sources can be added to SDP in a way that avoids signaling glare.

1.1.4. Interworking with other devices

When interacting with devices that do not apply all of the techniques described in this document, it must be possible to degrade gracefully to a usable basic experience. At a minimum, this basic experience should support setting up one audio stream and more than one video stream with existing videoconferencing equipment designed to establish a small number of simultaneous audio and video flows. For the remainder of this document, we will call these devices "legacy devices," although it should be understood that statements about legacy devices apply equally to future devices that elect not to use the techniques described in this document.

1.1.5. Avoidance of excessive port allocation

When there are dozens or hundreds of streams, it is desirable to avoid creating dozens or hundreds of transports, as empirical data shows a clear inverse relationship between number of transports (NAT bindings) and call success rate. While BUNDLE helps avoid creating large numbers of transports, it is also desirable to avoid creating large numbers of ports during call setup.

1.1.6. Simple binding of MediaStreamTrack to SDP

In WebRTC, each media source is identified by a MediaStreamTrack object. In order to ensure that the MSTs created by the sender show up at the receiver, each MST's id attribute needs to be reflected in SDP.

1.1.7. Support for RTX, FEC, simulcast, layered coding

For robust applications, techniques like RTX and FEC are used to protect media, and simulcast/layered coding can be used to provide support to heterogeneous receivers. It needs to be possible to support these techniques, allow the recipient to optionally use or not use them on a source-by-source basis; and for simulcast/layered scenarios, to control which simulcast streams or layers are received.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This draft uses the API and terminology described in [webrtc-api].

5-tuple: A collection of the following values: source IP address, source transport port, destination IP address, destination transport port and transport protocol.

Transport-Flow: An transport 5 Tuple representing the UDP source and destination IP address and port over which RTP is flowing.

m-line: An SDP [RFC4566] media description identifier that starts with an "m=" field and conveys the following values: media type, transport port, transport protocol and media format descriptions.

Offer: An [RFC3264] SDP message generated by the participant who wishes to initiate a multimedia communication session. An Offer describes the participant's capabilities for engaging in a multimedia session.

Answer: An [RFC3264] SDP message generated by the participant in response to an Offer. An Answer describes the participant's capabilities in continuing with the multimedia session with in the constraints of the Offer.

This draft avoids using terms that implementors do not have a clear idea of exactly what they are - for example RTP Session.

1.3. Syntax Conventions

The SDP examples given in this document deviate from actual on-the-wire SDP notation in several ways. This is done to facilitate readability and to conform to the restrictions imposed by the RFC formatting rules. These deviations are as follows:

- Any line that is indented (compared to the initial line in the SDP block) is a continuation of the preceding line. The line break and indent are to be interpreted as a single space character.

- Empty lines in any SDP example are inserted to make functional divisions in the SDP clearer, and are not actually part of the SDP syntax.

- Excepting the above two conventions, line endings are to be interpreted as <CR><LF> pairs (that is, an ASCII 13 followed by an ASCII 10).

- Any text starting with the string "//" to the end of the line is inserted for the benefit of the reader, and is not actually part of the SDP syntax.

2. Solution Overview

At a high level, the solution described in this document can be summarized as follows:

1. Each media stream track is represented by its own unique m-line. This is a strict one-to-one mapping; a single media stream track cannot be spread across several m-lines, nor may a single m-line represent multiple media stream tracks. Note that this requires a modification to the way simulcast is currently defined by the individual draft [I-D.westerlund-avtcore-rtp-simulcast]. This does not preclude "application level" simulcasting; i.e., the creation of multiple media stream tracks from a single source.

2. Each m-line is marked with an a=ssrc attribute to correlate it with its RTP packets. Absent any other signaled extension, multiple SSRCs in a single m-line are interpreted as alternate

sources for the same media stream track: although senders can switch between the SSRCs as frequently as desired, only one should be sent at any given time.

3. Each m-line contains an MSID value to correlate it with a Media Stream ID and the Media Stream Track ID.

4. To minimize port allocation during a call, we rely on the BUNDLE [I-D.ietf-mmusic-sdp-bundle-negotiation] mechanism.

5. To reduce port allocation during call set-up, applications can mark less-critical media stream tracks in such a way that they will not require any port allocation, with the resulting property that such streams only work in the presence of the BUNDLE mechanism.

6. To address glare, we define a procedure via which partial offer/answer exchanges may take place. These exchanges operate on a single m-line at a time, rather than an entire SDP body. These operations are defined in a way that can completely avoid glare for stream additions and removals, and which reduces the chance of glare for changes to active streams. This approach requires all m-lines to contain an a=mid attribute.

7. All sources in a single bundle are required to contain identical attributes except for those that apply directly to a media stream track (such as label, msid, and resolution). See those attributes marked "IDENTICAL" in [I-D.nandakumar-mmusic-sdp-mux-attributes] for details.

8. RTP and RTCP streams are demultiplexed strictly based on their SSRC. However, to handle legacy cases and signaling/media races, correlation of streams to m-sections can use other mechanisms, as described in Section 3.2.

3. Detailed Description

3.1. Bundle-Only M-Lines

Even with the use of BUNDLE, it is expensive to allocate ICE candidates for a large number of m-lines. An offer can contain "bundle-only" m-lines which will be negotiated only by endpoints which implement this specification and ignored by other endpoints.

OPEN ISSUE: While it's probably pretty clear that this behavior will be controlled, in WebRTC, via a constraint, the "default" behavior -- that is, whether a line is "bundle-only" when there is no constraint present -- needs to be settled. This is a balancing

act between maximizing interoperation with legacy equipment by default or minimizing port use during call setup by default.

In order to offer such an m-line, the offerer does two things:

- Sets the port in the m-line to 0. This indicates to old endpoints that the m-line is not to be negotiated.

- Adds an a=bundle-only line. This indicates to new endpoints that the m-line is to be negotiated if (and only if) bundling is used.

An example offer that uses this feature looks like this:

```
v=0
o=- 20518 0 IN IP4 203.0.113.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=group:BUNDLE S1 S2 S3
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
        42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7

m=audio 54400 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 20970
a=mid:1
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=ssrc:53280
a=candidate:0 1 UDP 2113667327 203.0.113.1 54400 typ host
a=candidate:1 2 UDP 2113667326 203.0.113.1 54401 typ host

m=video 0 RTP/SAVPF 96 97
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 1714
a=mid:2
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
a=rtcp-mux
a=ssrc:49152
a=bundle-only
```

```
m=video 0 RTP/SAVPF 96 97
a=msid:ma tc
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 57067
a=mid:3
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
a=rtcp-mux
a=ssrc:32768
a=bundle-only
```

An old endpoint simply rejects the bundle-only m-lines by responding with a 0 port. (This isn't a normative statement, just a description of the way the older endpoints are expected to act.)

```
v=0
o=- 20518 0 IN IP4 203.0.113.1
s=
t=0 0
c=IN IP4 203.0.113.2
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
        42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7

m=audio 55400 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=candidate:0 1 UDP 2113667327 203.0.113.2 55400 typ host
a=candidate:1 2 UDP 2113667326 203.0.113.2 55401 typ host

m=video 0 RTP/SAVPF 96 97
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv

m=video 0 RTP/SAVPF 96 97
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
```

A new endpoint accepts the m-lines (both bundle-only and regular) by offering m-lines with a valid port, though this port may be duplicated as specified in Section 6 of [I-D.ietf-mmusic-sdp-bundle-negotiation]. For instance:

```
v=0
o=- 20518 0 IN IP4 203.0.113.2
s=
t=0 0
c=IN IP4 203.0.113.2
a=group:BUNDLE B1 B2 B3
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
        42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7

m=audio 55400 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 24860
a=mid:1
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=ssrc:35987
a=candidate:0 1 UDP 2113667327 203.0.113.2 55400 typ host

m=video 55400 RTP/SAVPF 96 97
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 49811
a=mid:B2
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
a=rtcp-mux
a=ssrc:9587
a=bundle-only

m=video 55400 RTP/SAVPF 96 97
a=msid:ma tc
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 9307
a=mid:3
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
```

```
a=rtcp-mux
a=ssrc:21389
a=bundle-only
```

Endpoints MUST NOT accept bundle-only m-lines if they are not part of an accepted bundle group.

3.2. Correlation

The system under consideration has three constructs the need to be mutually correlated for proper functioning: m-lines, media stream tracks, and RTP sources. These correlations are described in the following sections.

3.2.1. Correlating RTP Sources with m-lines

Sending several media streams over a single transport 5-tuple can pose challenges in the form of stream identification and correlation. This proposal maintains the use of SSRC as the single demultiplexing point for multiple streams sent between a transport 5-tuple.

Nominally, this correlation is performed by including a=ssrc attributes in the SDP. Under ideal circumstances, the use of a=ssrc in the SDP exchanged between endpoints is sufficient to correlate a demultiplexed stream to its m-line. However, at least three unrelated situations can arise that make correlation using an alternate mechanism advantageous.

During call establishment, circumstances may arise under which an endpoint can send an offer for a new stream, and begin receiving that media stream prior to receiving the SDP that correlates its SSRC to the m-line. For such cases, the endpoint will not know how to handle the media, and will most probably be forced to discard it. This can lead to media stream "clipping," which has a strongly negative impact on user experience. For audio streams, an the "hello" of the answering party can be lost; for video streams, the initial I-frame can be lost, leading to corrupted or missing video until another I-frame is sent.

In the rare circumstance that a SSRC change for an existing media source is required, then any party that has changed its SSRC needs to inform the remote participants of the updated mapping, e.g. via a new SDP offer. Since any media sent with the new SSRC cannot be rendered until the new offer/answer exchange takes place, the clipping concern mentioned above exists here as well.

A different problem can arise when interoperating with legacy equipment. A number of circumstances can lead to the inability of a legacy endpoint to include SSRC information in its SDP. For example, in a system that decomposes signaling and media into different network devices, the protocol used to communicate between the boxes frequently will not include SSRC information, making it impossible to include in the SDP. If these devices choose to implement bundling, correlation of media streams to m-lines requires an alternate correlator.

These cases (and possibly other similar situations) can be ameliorated by using information in the media stream itself as a correlator to the SDP offer. If a packet arrives with an SSRC that is not yet associated with an m-line, we would ideally have some means of correlating it prior to the arrival of the answer.

The authors reiterate and emphasize that this technique is used solely for the purposes of correlation of an RTP stream to an SDP m-line after that stream has already been demultiplexed. Demultiplexing of multiple streams on a single transport address continues to be based on SSRC values.

3.2.1.1. RTP Header Extension Correlation

The preferred mechanism for such correlation is a new RTP header extension [RFC5285] that can be used near the beginning of an RTP stream to correlate RTP packets for which SSRC mapping information is not available. We propose that WebRTC implementations MUST implement this mechanism. We expect and that all other users of the BUNDLE extension SHOULD make use of it.

Although additional specification for this mechanism would be required for interoperability, the thumbnail sketch of such correlation is described below.

An implementation making use of this mechanism for local correlation includes an a=extmap attribute in the m-lines for which it wishes to use the mechanism. This attribute includes a mapping from the RTP header ID to the URL, as well as a 16-bit identifier (expressed as an integer) used for correlation; one such m-line would look like this:

```
m=audio 55400 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 7582  // NEW
a=mid:1
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
```

```
a=sendrecv
a=rtcp-mux
a=ssrc:35987
a=candidate:0 1 UDP 2113667327 203.0.113.2 55400 typ host
```

The remote endpoint, if it supports this extension, MUST include an RTP header extension in several (on the order of 3 to 10) of the initial RTP packets in the stream. The value of this header extension will contain the correlator from the extmap line (in the above example, 7582).

3.2.1.2. Payload Type Correlation

To support implementations that cannot implement the RTP header extension described in Section 3.2.1.1 but which wish to use the BUNDLE mechanism, we allow an alternate (but less-preferred) means of correlation using payload type. This approach takes advantage of the fact that the offer contains payload types chosen by its creator, which will be present in any RTP received from the remote party. If these payload types are unique, then they can be used to reliably correlate incoming RTP streams to their m= lines.

Because of its inherent limitations, it is advisable to use other correlation techniques than PT multiplexing if at all possible. In order to accomplish this, we propose, for WebRTC, that use of this technique be controlled by an additional constraint passed to createOffer by the Web application.

If this constraint is set, the browser MUST behave as described in this section. If the constraint is not set, the browser MUST use identical PTs for the same codec values within each m-line bundle.

When such a constraint is present, implementations attempt to entirely exhaust the dynamic payload type numbering space before re-using a payload type within the scope of a local transport address. If such a constraint is present and the payload type space would ordinarily be exhausted within the scope of a local transport address, the implementation MAY (at its discretion) take any of the following actions:

1. Bind to multiple local transport addresses (using different BUNDLE groups) for the purpose of keeping the {payload type, transport address} combination unique.

2. Signal a failure to the application.

OPEN ISSUE: The above text specifically calls out "dynamic payload type numbering space," which consists of payload types 96 through 127. This is the most conservative range of payload types possible, with the greatest chance of exhaustion in normal use. In practice, it may make more sense to use a different range. The canonical description of payload type allocation strategies for RTP/AVP and its related profiles is given in section 3 of [RFC3551]. Roughly summarized: all values from 0 to 127 can be dynamically bound to codecs; codes from 96 to 127 should be preferred, followed by previously unassigned values, followed by statically assigned values. This is, however, modified by [RFC5761], which effectively eliminates payload types 64 through 95. Given these constraints, reasonable proposals (in order of most conservative to most aggressive) would include:

1. The dynamic range (96-127), for 32 usable payload types. This is meant to accommodate the most naive implementation possible, which is only capable of dynamically binding payload types in the dynamic range. Although not supported by current specifications, such limitations are suspected to exist in some modern RTP libraries.

2. The dynamic range (96-127), followed by the contiguous unassigned range (35-63), for 61 usable payload types. This approach is intended to accommodate those implementations that do not support dynamic binding for payload types for which an "audio/video" type is registered in the IANA registry.

3. The dynamic range (96-127) followed by all unassigned payload types (20-24, 27, 29, 30, and 35-63), for 69 usable payload types. This approach is intended to accommodate those implementations that are incapable of re-binding statically assigned payload types, while making use of all other available values.

4. The dynamic range (96-127) followed by all unassigned payload types (20-24, 27, 29, 30, and 35-63), followed by the statically assigned payload types (0-19, 25, 26, 28, and 31-34) for 96 usable payload types. This approach is most consistent with current IETF specifications, but is expected to cause interoperability issues with existing implementations (including libraries currently in use in early WebRTC implementations).

Note that the presence or absence of the aforementioned flag does not affect how incoming streams are correlated: if the RTP header extension for correlation is present, it is used in preference to the payload type. Conversely, if the flag is absent, and the RTP

contains no such header, then the payload type may be used for correlation inasmuch as a media line can be unambiguously identified. Of course, if the SSRC information has been made available in SDP prior to a need for stream correlation, then it can also be used for this purpose.

3.2.2. Correlating Media Stream Tracks with m-lines

Media Stream Tracks IDs are correlated with M-Lines directly by including an MSID in each m-line. The MSID also provides the Media Stream ID. (Note the format of the MSID used here is slightly different than what was proposed in the current MSID draft as that draft assumed multiple tracks in a single m-line and this proposal moves to a solution where there is a one to one relation between the Track and MSID. This work assumes the MSID draft will be updated to match the syntax used user which simply provides the value of the MediaStream ID and MediaStreamTrack ID on an "a=msid" line.)

3.2.3. Correlating Media Stream Tracks with RTP Sources

Media Stream Tracks are correlated with RTP sources transitively through the RTP-Source <=> M-Line <=> Media-Stream-Track relationship. Since the Media-Stream-Track <=> M-Line binding is established in the SDP offer, and the M-Line <=> RTP-Source binding can be handled as described in Section 3.2.1, none of the previously identified issues arise.

3.3. Handling of Simulcast, Forward Error Correction, and Retransmission Streams

Simulcast refers to taking a single capture (e.g., a camera), and encoding it multiple times at different resolutions and / or frame rates. For example, a device with a single HD camera may send one version of the video at full HD resolution, and a second version encoded at a low resolution. This would allow a video conferencing bridge to be able to send the high resolution copy to some destination and low resolution copy to other destinations without having to recode the video at the conference bridge.

Forward Error Correction (FEC) and Retransmission (RTX) streams are techniques that can provide stream robustness in the face of packet loss. These approaches frequently make use of different payload types and different SSRC values than the stream to which they apply.

In cases where a media source needs to correspond to more than one RTP flow, e.g. RTX, FEC, or simulcast, the a=ssrc-group [RFC5576] concept is used to create a grouping of SSRCs for a single media stream track. Each SSRC is declared using a=ssrc attributes, the

same MSID is shared between the SSRCs, and the a=ssrc-group attribute defines the behavior of the grouped SSRCs.

These groupings are used to perform demux of the incoming RTP streams and associate them (by SSRC) with their primary flows (modulo the behavior described in Section 3.2.1, if applicable). This multiplexing of RTX and FEC in a single RTP session is already well-defined; RTX SSRC-multiplexing behavior is defined in [RFC4588], and FEC SSRC-multiplexing behavior is defined in [RFC5956].

Note that both RTC and FEC also include SDP expressions that use different m= lines for the correction streams (cf. [RFC4588], section 8.7 and [RFC5956], section 4.2). These formats intend for correlation of streams to be based on transport addresses, which is inapplicable for bundled media streams. Our specific proposal is: (1) bundling implementations will never generate such a format; and (2) bundling implementations MAY choose to accept SDP in such a format or MAY simply reject the repair streams and proceed as if the indicated repair format is not supported.

For multi-resolution simulcast, we can create a similar ssrc-group, and adapt the imageattr attribute defined in [RFC6236] for the a=ssrc line attribute to indicate the send resolution for a given simulcast stream. (This will be added to [I-D.westerlund-avtcore-rtp-simulcast], as outlined in Section 2, bullet 1). In the example below, the SDP advertises a simulcast of a camera source at two different resolutions, as well as a screen-share source that supports RTX; a=ssrc-group is used to correlate the different SSRCs as part of a single media source.

Note that a characteristic of this approach is that it does not allow for independently setting attributes for simulcast, FEC, and RTX streams aside from those in fmtp. In particular, attributes such as ptime and framerate are shared between the streams that are grouped together for a simulcast group.

```
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 imageattr:96 [x=1280,y=720]
a=ssrc:47182 imageattr:96 [x=640,y=360]
a=ssrc-group:SIMULCAST 29154 47182

m=video 0 RTP/SAVPF 96 97          // slide video
```

```
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 26267
a=mid:2
a=rtpmap:96 VP8/90000
a=rtpmap:97 rtx/90000
a=sendrecv
a=rtcp-mux
a=fmtp:97 apt=96;rtx-time=3000
a=ssrc:45982
a=ssrc:9827
a=ssrc-group:FID 45982 9827        // FID provides SSRC correlation
a=bundle-only
```

Providing explicit resolutions on a per-SSRC basis for SIMULCAST groupings allows an intermediary (such as a Media Translator [RFC5117]) to be able to select an appropriate SIMULCAST layer without inspecting the media stream, which could otherwise require decrypting and possibly partially decoding media packets.

3.4. Glare Minimization

To allow for guaranteed glareless addition and removal of streams, and to provide for a reduced chance of glare in stream attribute changes, we propose a technique that allows for m-lines to be changed independently of each other.

The proposal for doing so is performed using "partial offers" and "partial answers." Using this technique has two key prerequisites: (1) all offer/answer exchanges in the session have contained "a=mid" attributes [RFC5888] for each m-line, and (2) both sides are known to support the partial offer/answer technique (either because they are part of a single domain of control, or because use of this technique has been explicitly signaled).

The use of a partial SDP body will be explicitly signaled, e.g., using a different MIME type for SIP, or using a different "type" for the WebRTC API.

The authors recognize that further formal definition would be required to describe this technique. These are left as future study for the appropriate venues, such as the W3C WebRTC WG and the SIPCORE WG. As a thumbnail sketch: For WebRTC, we envision that we would add a new constraint to createOffer, requesting that a partial offer be generated (if possible). The resulting RTCSessionDescription would contain only the m-lines that have changed since the most recent offer/answer exchange, and would have a type of "partialOffer." When createAnswer is called after receipt of a partialOffer, it would

create a partialAnswer, containing only the m-lines referenced in the partial offer, that can be provided to the remote party.

3.4.1. Adding a Stream

To add a stream glarelessly, a party creates a "partial offer" consisting of an m-line and all of its attributes. This m-line contains an mid that has not yet been used in the session. To reduce the chance of collision to effectively zero, this mid MUST contain at least 32 characters chosen randomly from full set of 79 characters allowed in a token. It then sends this partial offer to the remote party and awaits a partial answer.

Upon receipt of a partial offer, an implementation examines the mid in it. If the mid does not match any existing mid in the session, then it represents a new media stream. Assuming the recipient does not have an outstanding, unanswered partial offer that also adds a stream, this new m-line is simply appended to the end of the existing session description, the SDP version is incremented by one, and a partial answer is created. This partial answer consists of an m-line and its attributes, and has an mid matching the one from the partial offer.

If the recipient of a partial offer that contains a new mid has also sent a partial offer adding a new stream to the session, then ambiguity can arise regarding the canonical ordering of m-lines within the session. In this situation, both partial offer/answer exchanges are allowed to complete independently (as no fundamental data glare has occurred). However, the order in which they are appended to the session description is synchronized by performing a lexical comparison between each m-lines mid attribute: the m-line with the lexically smaller mid attribute is appended first, while the other m-line is appended after it.

3.4.2. Changing a Stream

Partial offers may also be generated for modification of an existing stream. In this case, the mid in the partial offer will match an existing mid in the session description.

Upon receipt of a partial offer, an implementation examines the mid in it. If the mid matches any existing mid in the session, then it represents a modification to that m-line. Assuming the recipient does not have an outstanding, unanswered partial offer that also modifies that exact same stream, this m-line is treated as an independent renegotiation of that stream (only). The SDP version is incremented by one, and a partial answer is created. This partial answer consists of an m-line and its attributes, and has an mid matching the one from the partial offer.

If the recipient of a partial offer that contains an existing mid has also sent a partial offer to change that exact same stream, and neither the received nor the sent partial offer contains an "a=inactive" attribute, then a legitimate glare condition has arisen. Normal glare recovery procedures -- e.g., using a tie-breaker token or a back-off timer -- must be engaged to resolve the conflict.

3.4.3. Removing a Stream

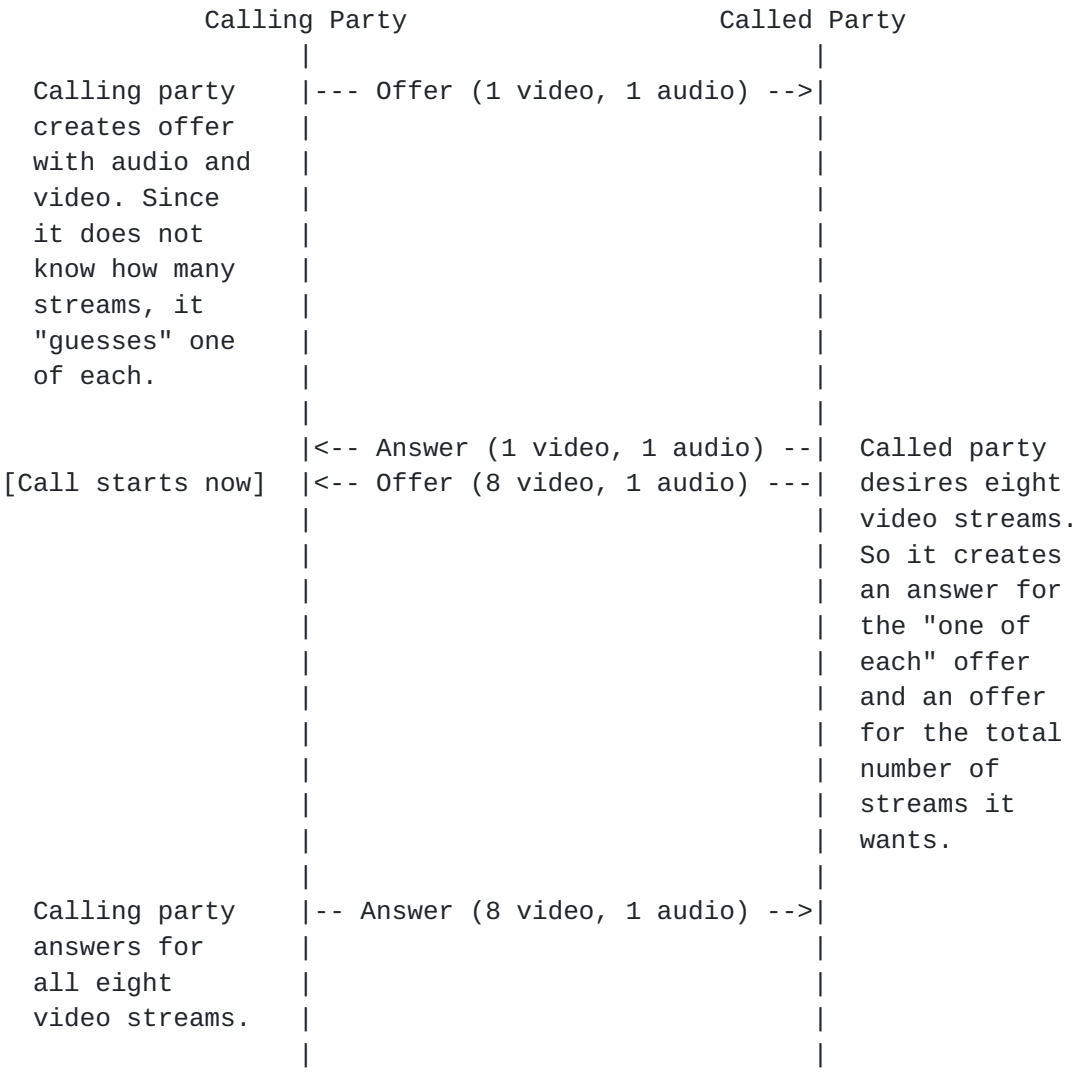
To remove a stream in a way that eliminates the chance of glare, an implementation generates a new partial offer, with an mid matching the m-line it wants to remove. This partial offer contains an a=inactive attribute, indicating that the stream is being deactivated.

If the recipient of a partial offer that contains an existing mid has also sent a partial offer to change that exact same stream, and either one of the received or the sent partial offer contains an "a=inactive" attribute, then a the stream is deactivated. At this point, both partial offers are discarded, the corresponding m-line in the session is modified by changing any a=sendonly, a=recvonly, or a=sendrecv attribute to a=inactive (or, if no such attributes are present, an a=inactive attribute is added), and a partial answer is generated representing this single change.

3.5. Negotiation of Stream Ordinality

Within advanced applications, circumstances can easily arise in which the party creating the offer does not know ahead of time the number of streams the remote party will desire. For example, in a meet-me videoconference application that sends a separate stream for each participant, a client creating an offer to send to the conference focus does not necessarily know how many video streams to indicate in its SDP. Although this can be potentially be solved in an application-specific way (e.g., by always offering the maximum number of streams known to be supported by the application), this is not always desirable or even possible.

To address this situation, a three-way handshake can be employed.



The first leg of this handshake consists of an offer sent by the calling party. This offer contains at least one m-line for each type of media the offerer wishes to use in the session. The authors draw special attention to the clause "at least" in the preceding sentence: offerers can use external knowledge, hinting, or simple guesses to offer additional m-lines.

Upon receipt of such an offer, the called party examines the number of streams of each media type being requested. If the number of streams is equal to or greater than the number of total streams that the called party desires at this time, it simply forms an answer to complete the offer/answer exchange [RFC3264], and the call is set up.

On the other hand, if the called party determines that more streams are necessary than are indicated in the initial offer, it responds by first creating an answer with the same number of streams as were present in the initial offer. It additionally creates a new answer that contains the number of streams it desires. This answer/offer pair is sent to the calling party, in a single message if supported by the signaling protocol (as will frequently be the case for WebRTC), or in two consecutive messages in a way that guarantees in-order delivery.

When the calling party receives this answer, it establishes the session, and all of the streams that were negotiated in this first offer/answer exchange. So, within a single signaling round trip, the initial set of streams (consisting of those the calling party included in its initial offer) are established.

When the calling party receives the subsequent offer, it comprises the beginning of a completely new RFC 3264 offer/answer exchange [RFC3264]. The calling party creates an answer that fully describes all of the streams in the session, and sends it to the called party. Consequently, within 1.5 round trips, the entire call is set up and all associated streams can be sent and received.

Of particular note is the fact that this model does not deviate from normal RFC3264 offer/answer handling, even when three-way handshaking is necessary.

3.6. Compatibility with Legacy uses

Due to the fact that this approach re-uses existing SDP constructs for indicating parameters in a media section, it remains compatible with legacy clients. Of particular note is the handling of "bundle-only" media sections, described in Section 3.1. Offers generated by an RTCWEB client and sent to a legacy client will simply negotiate those media the RTCWEB client did not use the "bundle-only" extension with. This allows RTCWEB clients to select which media streams are important for interoperability with legacy clients (by not making them bundle-only), and which ones are not. Offers generated by legacy clients will simply omit any bundle-related attributes, and the RTCWEB client will be able to process the SDP otherwise identically to the SDP received from RTCWEB clients: each m-line represents a different media stream, and contains a description of that stream in a syntax identical to the syntax used between RTCWEB clients.

With the bundle-only approach, only those streams that are "important for interoperability" will require allocation of ports and ICE exchanges. By doing so, working with non-multiplexing clients is

enabled without requiring excess resource allocation for those streams that are not critical for proper user experience.

4. Examples

In all of these examples, there are many lines that are wrapped due to column width limitation. It should be understood these lines are not wrapped in the real SDP.

The convention used for IP addresses in this drafts is that private IP behind a NAT come from 192.0.2.0/24, the public side of a NAT comes from 198.51.100.0/24 and the TURN servers have addresses from 203.0.113.0/24. Typically the offer has an IP ending in .1 and the answer has an IP ending in .2.

The examples do not include all the parts of SDP that are used in RTCWeb (See [I-D.ietf-rtcweb-rtp-usage]) as that makes the example unwieldy to read but instead focuses on showing the parts that are key for the multiplexing.

4.1. Simple example with one audio and one video

The following SDP shows an offer that offers one audio stream and one video steam with both a STUN and TURN address. It also shows unique payload across the audio and video m=lines for the Answerer that does not support BUNDLE semantics.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:074c6550
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068
a=fingerprint:sha-1
        99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07
a=group:BUNDLE m1 m2

m=audio 56600 RTP/SAVPF 0 109
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 33424
a=mid:m1
a=ssrc:53280
a=rtpmap:0 PCMU/8000
a=rtpmap:109 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
```

```
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
        192.0.2.1 rport 54400
a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
        192.0.2.1 rport 54401
a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
        192.0.2.1 rport 54400
a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
        192.0.2.1 rport 54401

m=video 56602 RTP/SAVPF 99 120
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 35969
a=mid:m2
a=ssrc:49843
a=rtpmap:99 H264/90000
a=fmtp:99 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:120 VP8/90000
a=sendrecv
a=rtcp-mux
a=candidate:3 1 UDP 2113667327 192.0.2.1 54402 typ host
a=candidate:4 2 UDP 2113667326 192.0.2.1 54403 typ host
a=candidate:3 1 UDP 694302207 198.51.100.1 55502 typ srflx raddr
        192.0.2.1 rport 54402
a=candidate:4 2 UDP 169430220 198.51.100.1 55503 typ srflx raddr
        192.0.2.1 rport 54403
a=candidate:3 1 UDP 73545215 203.0.113.1 56602 typ relay raddr
        192.0.2.1 rport 54402
a=candidate:4 2 UDP 51989708 203.0.113.1 56603 typ relay raddr
        192.0.2.1 rport 54403
```

The following shows an answer to the above offer from a device that does not support bundle or rtcp-mux.

```
v=0
o=- 16833 0 IN IP4 198.51.100.2
s=
t=0 0
c=IN IP4 203.0.113.2
a=ice-ufrag:c300d85b
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2
a=fingerprint:sha-1
        91:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:03

m=audio 60600 RTP/SAVPF 109
a=msid:ma ta
```

```
a=rtpmap:109 opus/48000
a=ptime:20
a=sendrecv
a=candidate:0 1 UDP 2113667327 192.0.2.2 60400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.2 60401 typ host
a=candidate:0 1 UDP 1694302207 198.51.100.2 60500 typ srflx raddr
        192.0.2.2 rport 60400
a=candidate:1 2 UDP 1694302206 198.51.100.2 60501 typ srflx raddr
        192.0.2.2 rport 60401
a=candidate:0 1 UDP 73545215 203.0.113.2 60600 typ relay raddr
        192.0.2.1 rport 60400
a=candidate:1 2 UDP 51989708 203.0.113.2 60601 typ relay raddr
        192.0.2.1 rport 60401

m=video 60602 RTP/SAVPF 99
a=msid:ma tb
a=rtpmap:99 H264/90000
a=fmtp:99 profile-level-id=4d0028;packetization-mode=1
a=sendrecv
a=candidate:2 1 UDP 2113667327 192.0.2.2 60402 typ host
a=candidate:3 2 UDP 2113667326 192.0.2.2 60403 typ host
a=candidate:2 1 UDP 694302207 198.51.100.2 60502 typ srflx raddr
        192.0.2.2 rport 60402
a=candidate:3 2 UDP 169430220 198.51.100.2 60503 typ srflx raddr
        192.0.2.2 rport 60403
a=candidate:2 1 UDP 73545215 203.0.113.2 60602 typ relay raddr
        192.0.2.2 rport 60402
a=candidate:3 2 UDP 51989708 203.0.113.2 60603 typ relay raddr
        192.0.2.2 rport 60403
```

The following shows answer to the above offer from a device that does support bundle.

```
v=0
o=- 16833 0 IN IP4 198.51.100.2
s=
t=0 0
c=IN IP4 203.0.113.2
a=ice-ufrag:c300d85b
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2
a=fingerprint:sha-1
        91:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:03
a=group:BUNDLE m1 m2

m=audio 60600 RTP/SAVPF 109
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 39829
```

```
a=mid:m1
a=ssrc:35856
a=rtpmap:109 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=candidate:0 1 UDP 2113667327 192.0.2.2 60400 typ host
a=candidate:0 1 UDP 1694302207 198.51.100.2 60500 typ srflx raddr
        192.0.2.2 rport 60400
a=candidate:0 1 UDP 73545215 203.0.113.2  60600 typ relay raddr
        192.0.2.1 rport 60400

m=video 60600 RTP/SAVPF 99
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 45163
a=mid:m2
a=ssrc:2638
a=rtpmap:99 H264/90000
a=fmtp:99 profile-level-id=4d0028;packetization-mode=1
a=sendrecv
a=rtcp-mux
a=candidate:3 1 UDP 2113667327 192.0.2.2 60400 typ host
a=candidate:3 1 UDP 694302207 198.51.100.2 60500 typ srflx raddr
        192.0.2.2 rport 60400
a=candidate:3 1 UDP 73545215 203.0.113.2  60600 typ relay raddr
        192.0.2.2 rport 60400
```

4.2. Multiple Videos

Simple example showing an offer with one audio stream and two video streams.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
        42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
a=group:BUNDLE m1 m2 m3

m=audio 56600 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 47434
a=mid:m1
```

```
   a=ssrc:32385
   a=rtpmap:0 PCMU/8000
   a=rtpmap:96 opus/48000
   a=ptime:20
   a=sendrecv
   a=rtcp-mux
   a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
   a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
   a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
           192.0.2.1 rport 54400
   a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
           192.0.2.1 rport 54401
   a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
           192.0.2.1 rport 54400
   a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
           192.0.2.1 rport 54401

   m=video 56602 RTP/SAVPF 96 98
   a=msid:ma tb
   a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 22705
   a=mid:m2
   a=ssrc:43985
   a=rtpmap:96 H264/90000
   a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
   a=rtpmap:98 VP8/90000
   a=sendrecv
   a=rtcp-mux
   a=candidate:2 1 UDP 2113667327 192.0.2.1 54402 typ host
   a=candidate:3 2 UDP 2113667326 192.0.2.1 54403 typ host
   a=candidate:2 1 UDP 694302207 198.51.100.1 55502 typ srflx raddr
           192.0.2.1 rport 54402
   a=candidate:3 2 UDP 169430220 198.51.100.1 55503 typ srflx raddr
           192.0.2.1 rport 54403
   a=candidate:2 1 UDP 73545215 203.0.113.1 56602 typ relay raddr
           192.0.2.1 rport 54402
   a=candidate:3 2 UDP 51989708 203.0.113.1 56603 typ relay raddr
           192.0.2.1 rport 54403
   a=ssrc:11111 cname:45:5f:fe:cb:81:e9

   m=video 56604 RTP/SAVPF 96 98
   a=msid:ma tc
   a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 64870
   a=mid:m3
   a=ssrc:54269
   a=rtpmap:96 H264/90000
   a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
   a=rtpmap:98 VP8/90000
   a=sendrecv
```

```
a=rtcp-mux
a=candidate:4 1 UDP 2113667327 192.0.2.1 54404 typ host
a=candidate:5 2 UDP 2113667326 192.0.2.1 54405 typ host
a=candidate:4 1 UDP 694302207 198.51.100.1 55504 typ srflx raddr
        192.0.2.1 rport 54404
a=candidate:5 2 UDP 169430220 198.51.100.1 55505 typ srflx raddr
        192.0.2.1 rport 54405
a=candidate:4 1 UDP 73545215 203.0.113.1 56604 typ relay raddr
        192.0.2.1 rport 54404
a=candidate:5 2 UDP 51989708 203.0.113.1 56605 typ relay raddr
        192.0.2.1 rport 54405
a=ssrc:22222 cname:45:5f:fe:cb:81:e9
```

4.3. Many Videos

This section adds three video streams and one audio. The video streams are sent in such a way that they they are only accepted if the far side supports bundle using the "bundle only" approach described in Section 3.1. The video streams also use the same payload types so it will not be possible to demux the video streams from each other without using the SSRC values.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
        42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
a=group:BUNDLE m0 m1 m2 m3

m=audio 56600 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 6614
a=mid:m0
a=ssrc:12359
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=ssrc:12359 cname:45:5f:fe:cb:81:e9
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
```

```
          192.0.2.1 rport 54400
   a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
          192.0.2.1 rport 54401
   a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
          192.0.2.1 rport 54400
   a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
          192.0.2.1 rport 54401

   m=video 0 RTP/SAVPF 96 98
   a=msid:ma tb
   a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 24147
   a=mid:m1
   a=ssrc:26989
   a=rtpmap:96 H264/90000
   a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
   a=rtpmap:98 VP8/90000
   a=sendrecv
   a=rtcp-mux
   a=bundle-only
   a=ssrc:26989 cname:45:5f:fe:cb:81:e9

   m=video 0 RTP/SAVPF 96 98
   a=msid:ma tc
   a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 33989
   a=mid:m2
   a=ssrc:32986
   a=rtpmap:96 H264/90000
   a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
   a=rtpmap:98 VP8/90000
   a=sendrecv
   a=rtcp-mux
   a=bundle-only
   a=ssrc:32986 cname:45:5f:fe:cb:81:e9

   m=video 0 RTP/SAVPF 96 98
   a=msid:ma td
   a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 61408
   a=mid:m3
   a=ssrc:46986
   a=rtpmap:96 H264/90000
   a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
   a=rtpmap:98 VP8/90000
   a=sendrecv
   a=rtcp-mux
   a=bundle-only
   a=ssrc:46986 cname:45:5f:fe:cb:81:e9
```

4.4. Multiple Videos with Simulcast

This section shows an offer with with audio and two video each of which can send it two resolutions as described in Section 3.3. One video stream supports VP8, while the other supports H.264. All the video is bundle-only. Note that the use of different codec-specific parameters causes two different payload types to be used.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
        42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
a=group:BUNDLE m0 m1 m2

m=audio 56600 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 31727
a=mid:m0
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
        192.0.2.1 rport 54400
a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
        192.0.2.1 rport 54401
a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
        192.0.2.1 rport 54400
a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
        192.0.2.1 rport 54401

m=video 0 RTP/SAVPF 96 100
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 41664
b=AS:1756
a=mid:m1
a=rtpmap:96 VP8/90000
a=ssrc-group:SIMULCAST 58949 28506
a=ssrc:58949 imageattr:96 [x=1280,y=720]
a=ssrc:28506 imageattr:96 [x=640,y=480]
```

```
a=sendrecv
a=rtcp-mux
a=bundle-only

m=video 0 RTP/SAVPF 96 100
a=msid:ma tc
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 14460
b=AS:1756
a=mid:m2
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1;max-fr=30
a=rtpmap:100 H264/90000
a=fmtp:100 profile-level-id=4d0028;packetization-mode=1;max-fr=15
a=ssrc-group:SIMULCAST 18875 54986
a=ssrc:18875
a=ssrc:54986
a=sendrecv
a=rtcp-mux
a=bundle-only
```

4.5. Video with Simulcast and RTX

This section shows an SDP offer that has an audio and a single video stream. The video stream that is simulcast at two resolutions and has [RFC4588] style re-transmission flows.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
        42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
a=group:BUNDLE m0 m1

m=audio 56600 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 42123
a=mid:m0
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
```

```
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
        192.0.2.1 rport 54400
a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
        192.0.2.1 rport 54401
a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
        192.0.2.1 rport 54400
a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
        192.0.2.1 rport 54401

m=video 0 RTP/SAVPF 96 101
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 60725
b=AS:2500
a=mid:m1
a=rtpmap:96 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=96;rtx-time=3000
a=ssrc-group:SIMULCAST 78909 43567
a=ssrc-group:FID 78909 56789
a=ssrc-group:FID 43567 13098
a=ssrc:78909
a=ssrc:43567
a=ssrc:13098
a=ssrc:56789
a=sendrecv
a=rtcp-mux
a=bundle-only
```

4.6. Video with Simulcast and FEC

This section shows an SDP offer that has an audio and a single video stream. The video stream that is simulcast at two resolutions and has [RFC5956] style FEC flows.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
        42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
a=group:BUNDLE m0 m1
```

```
m=audio 56600 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 42123
a=mid:m0
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
        192.0.2.1 rport 54400
a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
        192.0.2.1 rport 54401
a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
        192.0.2.1 rport 54400
a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
        192.0.2.1 rport 54401

m=video 0 RTP/SAVPF 96 101
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 60725
b=AS:2500
a=mid:m1
a=rtpmap:96 VP8/90000
a=rtpmap:101 1d-interleaved-parityfec/90000
a=fmtp:96 max-fr=30;max-fs=8040
a=fmtp:101 L=5; D=10; repair-window=200000
a=ssrc-group:SIMULCAST 56780 34511
a=ssrc-group:FEC-FR 56780 48675
a=ssrc-group:FEC-FR 34511 21567
a=ssrc:56780
a=ssrc:34511
a=ssrc:21567
a=ssrc:48675
a=sendrecv
a=rtcp-mux
a=bundle-only
```

4.7. Video with Layered Coding

This section shows an SDP offer that has an audio and a single video stream. The video stream that is layered coding at 3 different resolutions based on [RFC5583]. The video m=lines shows 3 streams with last stream (payload 100) dependent on streams with payload 96 and 97 for decoding.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
        42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
a=group:BUNDLE m0 m1

m=audio 56600 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 42123
a=mid:m0
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
        192.0.2.1 rport 54400
a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
        192.0.2.1 rport 54401
a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
        192.0.2.1 rport 54400
a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
        192.0.2.1 rport 54401

m=video 0 RTP/SAVPF 96 97 100
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 60725
b=AS:2500
a=mid:m1
a=rtpmap:96 H264/90000
a=fmtp:96 max-fr=30;max-fs=8040
a=rtpmap:97 H264/90000
a=fmtp:97 max-fr=15;max-fs=1200
a=rtpmap:100 H264-SVC/90000
a=fmtp:100 max-fr=30;max-fs=8040
```

```
a=depend:100 lay m1:96,97;
a=ssrc:48970
a=ssrc:90898
a=ssrc:66997
a=sendrecv
a=rtcp-mux
a=bundle-only
```

5. Security Considerations

TBD

6. IANA Considerations

TBD

7. Acknowledgements

Thanks to Cullen Jennings and Suhas Nandakumar for their assistance in generating the SDP examples in this document.

Some of the material in this document was taken from [I-D.jennings-rtcweb-plan].

8. References

8.1. Normative References

[I-D.ietf-mmusic-sdp-bundle-negotiation]
Holmberg, C., Alvestrand, H., and C. Jennings, "Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers", draft-ietf-mmusic-sdp-bundle-negotiation-03 (work in progress), February 2013.

[I-D.jennings-mmusic-media-req]
Jennings, C., Uberti, J., and E. Rescorla, "Requirements from various WG for MMUSIC", draft-jennings-mmusic-media-req-00 (work in progress), February 2013.

[I-D.nandakumar-mmusic-sdp-mux-attributes]
Nandakumar, S., "A Framework for SDP Attributes when Multiplexing", draft-nandakumar-mmusic-sdp-mux-attributes-02 (work in progress), April 2013.

[I-D.westerlund-avtcore-rtp-simulcast]

Westerlund, M., Lindqvist, M., and F. Jansson, "Using Simulcast in RTP Sessions", draft-westerlund-avtcore-rtp-simulcast-02 (work in progress), February 2013.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC5576] Lennox, J., Ott, J., and T. Schierl, "Source-Specific Media Attributes in the Session Description Protocol (SDP)", RFC 5576, June 2009.

[RFC5888] Camarillo, G. and H. Schulzrinne, "The Session Description Protocol (SDP) Grouping Framework", RFC 5888, June 2010.

8.2. Informative References

[I-D.ietf-rtcweb-rtp-usage]
Perkins, C., Westerlund, M., and J. Ott, "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP", draft-ietf-rtcweb-rtp-usage-06 (work in progress), February 2013.

[I-D.jennings-rtcweb-plan]
Jennings, C., "Proposed Plan for Usage of SDP and RTP", draft-jennings-rtcweb-plan-01 (work in progress), February 2013.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.

[RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, July 2006.

[RFC5117] Westerlund, M. and S. Wenger, "RTP Topologies", RFC 5117, January 2008.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.

[RFC5583] Schierl, T. and S. Wenger, "Signaling Media Decoding Dependency in the Session Description Protocol (SDP)", RFC 5583, July 2009.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010.

[RFC5956] Begen, A., "Forward Error Correction Grouping Semantics in the Session Description Protocol", RFC 5956, September 2010.

[iana.rtp-pt]
IANA, "RTP Payload types (PT) for standard audio and video encodings", July 2013.

Available at http://www.iana.org/assignments/rtp-parameters/rtp-parameters.xhtml#rtp-parameters-1

[webrtc-api]
Bergkvist, Burnett, Jennings, Narayanan, , "WebRTC 1.0: Real-time Communication Between Browsers", October 2011.

Available at http://dev.w3.org/2011/webrtc/editor/webrtc.html

Authors' Addresses

Adam Roach
Mozilla
Dallas, TX
US

Phone: +1 650 903 0800 x863
Email: adam@nostrum.com

Justin Uberti
Google
747 6th St. S
Kirkland, WA 98033
USA

Email: justin@uberti.name

Martin Thomson
Microsoft
3210 Porter Drive
Palo Alto, CA 94304
US

Phone: +1 650 353 1925
Email: martin.thomson@skype.net