Zulu No/One Way Audio

- Bypassing the Zulu proxy
- STUN and TURN
  - STUN Connectivity
  - TURN Connectivity
  - STUN and TURN usage in Zulu and Asterisk
    - Asterisk
    - Zulu
- Session Description Protocol (SDP)
  - Session description
  - Time Description
  - Media description
- The Connection Process
  - HOST Candidates
  - SRFLX Candidates
  - RELAY Candidates
- Connectivity Failure
  - ICE Candidate Issues
    - Host Only
    - SRFLX (STUN) Candidates
    - RELAY (TURN) Candidates
- Debugging the SDP
  - Zulu Client
    - Deconstructing the client SDP lines
      - Global Lines
      - Audio Lines
        - ICE Candidates
        - ICE Parameters
        - DTLS Parameters
      - Audio Lines (Continued)
        - Codec Parameters
        - SSRC Parameters
  - Asterisk
    - Deconstructing Asterisk's SDP lines
      - Global Lines
      - Audio Lines
        - DTLS Parameters
        - ICE Parameters
        - ICE Candidates
        - Codec Parameters
        - Audio Lines (Continued)
- Putting it all Together
  - Debugging No/One Way Calls
    - Example #1: Initiate a Call From Zulu
      - Zulu Client
      - Asterisk
      - What went wrong?
      - Setting a STUN Server
      - Setting ICE Blacklist
      - Add ICE Host Candidates Maps
- Tools.Tests
- Reference

Note: if adding the stun server address in 'asterisk sip settings' under 'webrtc settings' & 'media transport settings', please restart the asterisk (fwconsole restart).

The Zulu desktop and mobile softphone utilize:

- The WebSocket Protocol as a Transport for the Session Initiation Protocol (SIP), RFC7118
- Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols, RFC5245

Zulu Signaling is also proxied over the Zulu port of 8002, this allows your server to only have to use 1 port* for all of zulu

This article is LARGELY copied from https://medium.com/xirsys/webrtc-connectivity-woes-and-you-dffb59f0d582

I can't possible say it better than Lee Sylvester and I thank him for his article in helping me to learn how to diagnose WebRTC

Bypassing the Zulu proxy
By default a Zulu client will use the one port (8002 by default) for all traffic - this includes the clients SIP signaling which is proxied via this same port. Bypassing the Zulu proxy only helps with the debugging process. If you are having No/One way audio it is never the result of the Zulu proxy. That is because the Zulu proxy is only utilized for SIP Signaling. However, when you are about to debug SIP packets through Asterisk on a busy system you will shortly realize that it's near impossible to differentiate the SIP packets because they all come from and go to the same source address: 127.0.0.1.

Changing this setting to "Yes" tells the Zulu client to connect to Asterisk directly, bypassing the Zulu proxy, you can then set logging through Asterisk for specifically your IP address.

Long term this should not stay "Yes", however there is no immediate problem if it is other than the fact that you will have to open another port (this port can be derived from the "HTTPS Bind Port" defined for the "Asterisk Builtin mini-HTTP server" in advanced settings)

**Important remark:**

In order to see this setting you will need to change the GUI to developer mode through the advanced settings.

Then, you will be able to change the proxy bypassing through the user management module.

The difference are merely where the sip signaling packets come from or are sent to.

**Proxied Zulu:**

```plaintext
<--- Received SIP request (3679 bytes) from WSS:127.0.0.1:56464 --->
INVITE sip:*43@myhost.com SIP/2.0
<snip>
v=0
o= 4365490366822564799 2 IN IP4 127.0.0.1
s=-
t=0 0
<snip>
```

**Unproxied Zulu:**

```plaintext
<--- Received SIP request (3679 bytes) from WSS:<client address>:56464 --->
INVITE sip:*43@myhost.com SIP/2.0
<snip>

v=0
do= 4365490366822564799 2 IN IP4 <client address>
s=-
t=0 0
<snip>
```

**STUN and TURN**
To debug Zulu, it first pays to understand a little of what STUN and TURN actually do.

If you don't already know, a STUN server allows clients to find out their public IP addresses, the type of NAT they are behind and the Internet side port associated by the NAT with a particular local port.

- STUN enables a device to find out its public IP address and the type of NAT service it's sitting behind.
- STUN operates on TCP and UDP port 3478.

While a TURN server is a network entity in charge of relaying media in VoIP related protocols

**STUN Connectivity**

https://medium.com/xirsys/webRTC-connectivity-woes-and-you-dffe89f0d582

When one machine wishes to establish a peer-to-peer connection with another machine on a different network, it first needs to trade public IP's. Kind of like swapping your phone number with a friend. However, machines on a network are usually only privy to knowing their local IP, which will look something like 192.168.x.x. This is no good for connectivity, because the machines are on separate networks, so those IPs are useless.

You can't phone your friend if all you have is her telephone extension number, but not the actual outside telephone number. When your machine makes requests to the internet, it sends a packet to a server via your NAT or router, including your local IP in the header. It is the job of the NAT to translate your local IP to a public IP, thereby updating the packet header. The packet then continues its journey to the server. When the server responds back to your computer, its packet will also contain your public IP in the header, but it gets translated to your local IP when it reaches your NAT, before being passed to you. Thus, your machine never gains access to the public IP. The STUN server works in the same way, by acting as an Echo Server. However, in order to inform you of your public IP, it first copies the IP from the header into the packet body, which your NAT will leave intact. When you receive the packet, you will know your public IP and can share this with the peer.

**TURN Connectivity**

https://medium.com/xirsys/webRTC-connectivity-woes-and-you-dffe89f0d582

TURN is an extension of STUN. It uses the same packet protocols, but extends it with additional features, enabling you to establish a server session for both you and the peer, so you can pass data to each other through the server. This is a necessary process if, for some reason, a direct peer-to-peer connection cannot be established between the two machines. The TURN connection has quite a complex handshake. In all connections with TURN, one person will be known as the client and the other person will be the peer. The client is the person who initiates the connection. This is important, because the client is responsible for sending all the necessary data and creating the link between the two machines. Only the client needs the authentication to use the TURN server. The client literally provides the access needed for the peer to connect, so the peer can be quite dumb about the process. Due to this ‘delegation of authority’, I have seen circumstances where TURN initiated in one direction may fail, however initiating in the other direction works.

**STUN and TURN usage in Zulu and Asterisk**

**Asterisk**

The settings for STUN AND TURN servers in Asterisk itself are blank by default. You need to configure them through Asterisk SIP Settings. But wait! Don't do that until AFTER we have figured out what the issue is. Adding STUN or TURN servers to Asterisk can have dire consequences if you don't know or understand what you are doing. Furthermore Asterisk is a powerful PBX engine and has many ways to configure/fix something for your network
Zulu

The settings for STUN and TURN servers for zulu clients are also set in Asterisk SIP Settings under the WebRTC Category. By default these appear blank however Zulu clients should always have a STUN server address defined and therefore if this parameter is left blank all Zulu clients will use the default of "http://stun.l.google.com:19302"

Session Description Protocol (SDP)

SDP stands for Session Description Protocol. SDP is used by Zulu to negotiate the session's parameters. SDP is a text based format. SDP is used for describing multimedia communication sessions for the purposes of session announcement, session invitation, and parameter negotiation. SDP does not deliver any media by itself but is used between endpoints for negotiation of media type, format, and all associated properties. The set of properties and parameters are often called a session profile.

Optional values are specified with *=* and each field must appear in the order shown below.

Session description
v= (protocol version number, currently only 0)
o= (originator and session identifier : username, id, version number, network address)
s= (session name : mandatory with at least one UTF-8-encoded character)
i=* (session title or short information)
u=* (URI of description)
e=* (zero or more email address with optional name of contacts)
p=* (zero or more phone number with optional name of contacts)
c=* (connection information—not required if included in all media)
b=* (zero or more bandwidth information lines)
One or more Time descriptions ("t=" and "r=" lines; see below)
z=* (time zone adjustments)
k=* (encryption key)
a=* (zero or more session attribute lines)
Zero or more Media descriptions (each one starting by an "m=" line; see below)

Time Description

t= (time the session is active)
r=* (zero or more repeat times)

Media description

m= (media name and transport address)
i=* (media title or information field)
c=* (connection information — optional if included at session level)
b=* (zero or more bandwidth information lines)
k=* (encryption key)
a=* (zero or more media attribute lines — overriding the Session attribute lines)

The Connection Process

In this example the word “Peer” and “Client” can be used interchangeably with Asterisk and the Zulu client. WebRTC, after all, is P2P.

With this knowledge in tow, let’s look at how the Zulu client establishes a connection with Asterisk. Zulu (through WebRTC) uses a specification called ICE to get a list of candidates that could be used for potential RTP communication. Connectivity information is shared with Asterisk via the SDP packets. Essentially, any means of getting a chunk of text data from client to peer and vice versa. The packets describing the connectivity options are called ICE Candidates, and they look a little like this:

a=candidate:Hae479ced 1 UDP 2130706431 192.168.1.4 10528 typ host

This particular ICE candidate shows the machine’s local IP and a ‘typ’ value of ‘host’. There are three different types of ICE candidate:
HOST Candidates

Host candidates are for same-network connections. If you’re connecting to a peer who is on the same network as you, and thus behind the same NAT, then you don’t need STUN or TURN. There’s no need for a public IP, because you’ll connect behind your NAT without having to pass through the public internet. Likewise, there’s no need for TURN, because there will be no problem seeing peers on your own network. It is for this reason that the host candidates are the first to be shared and the first connectivity to be attempted.

SRFLX Candidates

If the same-network connectivity fails, the client will attempt to make a STUN connection to acquire its public IP. STUN connections for Zulu are UDP only, so a failed attempt at this task will skip it entirely. If the client acquires its IP, it’ll send a SRFLX candidate to the peer. The peer will then repeat the process to acquire its IP, and if successful, will send that back to the client.

```
    a=candidate:1826845682 1 udp 1685921535 46.2.2.2 60883 typ srflx raddr 10.224.50.16 rport 60883 generation 0
```

If both client and peer manage to acquire their IPs and share them, this does not guarantee that peer-to-peer connectivity will work, only that it will be attempted.

RELAY Candidates

If peer-to-peer fails or if the STUN requests fail due to UDP packet blocking, the client will go ahead and make a TURN allocation with the TURN server. An allocation returns the public IP in its response, so this is the client’s second chance to receive this. Once it does, it sends relay candidates to the peer.

```
    a=candidate:2157334355 1 udp 33562367 180.6.6.6 54278 typ relay raddr 46.2.2.2 rport 38135 generation 0
```

Once relay candidates have been exchanged, TURN connectivity will be attempted and will, hopefully, succeed. For more information about ICE candidates and the SDP (Session Description Protocol) format in general, take a look at The Anatomy of a WebRTC SDP over at WebRTCHacks, for an interactive breakdown.

Connectivity Failure

Within the components of Zulu itself we’ve tried to add some handy features that allow you to know you are having issues specifically with WebRTC/Zulu audio. You can see an example of one such error above. In many common situations with audio (RTP) over WebRTC, Asterisk will not disconnect the call like it normally would with a normal SIP channel. This can give the illusion that it’s not an RTP issue, when in reality it is technically RTP but has more to do with ICE/WebRTC.

ICE Candidate Issues

Host Only
The first thing you'll notice is that it lists all of the ICE candidates generated during the call. You can open them all up and identify their 'typ' attribute, as noted previously in this article. Now, if all you see are HOST candidates, then your application was never able to reach out to our servers to help establish a connection. Sometimes, receiving only HOST candidates can mean that your network is restricting both the STUN UDP packets, as well as the TURN allocations over TCP (deep packet sniffing maybe?). However, as even I have encountered today, while running an internal project, HOST candidates can also indicate a coding issue. The quickest way to work this out is to run a WebRTC demo you know to work and to see what candidates that produces. Running our own demo internally quickly helped us work out that it was in fact a code issue that was to blame.

**SRFLX (STUN) Candidates**

If you see SRFLX candidates, then are you seeing them from both sides? In order to connect peer-to-peer, both client and peer need to send these. Receiving these on one side is not enough. What’s more, if you see SRFLX candidates and the connection is not occurring, why are you not also seeing RELAY candidates? Are you certain you’re using the correct ICE server string? Perhaps you forgot to swap out the public Google STUN server URL you were originally testing with?

**RELAY (TURN) Candidates**

So, your client is producing RELAY candidates. This is great. This means your client has at least sent Allocation packets to the TURN server and received a response, so it knows relay is an option. So, why isn’t it connecting? Does your network deep packet sniff and refuse UDP through TCP tunnelling or is it something else?

**Debugging the SDP**

While you normally might be used to running RTP set debug on/off in a WebRTC situation you’d be better off reading the SDP that is sent between Asterisk and the Zulu client.

**Zulu Client**

You can see the SDP in the Zulu client by `<needs to be defined>`

We can identify packets to be SDP because "Content-Type" is "application/sdp"

An example of a packet on Zulu with SDP looks like this:

```plaintext
INVITE sip:*43@myhost.com SIP/2.0
Via: SIP/2.0/WSS fo93pieho9ql.invalid;branch=z9hG4bK3346360
Max-Forwards: 70
To: <sip:*43@myhost.com>
From: <sip:904010@myhost.com>;tag=b0r948n56d
Call-ID: qlurm4d86pv7irde447p
CSeq: 7769 INVITE
Authorization: Digest algorithm=MD5, username="904010", realm="asterisk",
nonce="1529023675/0deca1542e814456a5557672ba6615984", uri="sip:*43@myhost.com",
response="4d9c39fe7744e4a8cf29ef6e8795d034", opaque="28cdd0e66897d6c8", qop=auth,
cnonce="d4n4qpcpcch8k", nc=00000001
Contact: <sip:iln2odb2@fo93pieho9ql.invalid;transport=wss;ob>
Allow: ACK,CANCEL,INVITE,MESSAGE,BYE,OPTIONS,INFO,NOTIFY,REFER
Supported: outbound
User-Agent: Zulu
Content-Type: application/sdp
Content-Length: 2867

v=0
o=- 2276445737850684162 2 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE audio
a=msid-semantic: WMS km0kdiSIIfNWE5sIKmB1V6vfJZnQtM6kfohJh
```
The important part here is the SDP itself, you'll notice if you don't focus specifically on just the SDP part that asterisk labels Via with "invalid". This is all part of how WebRTC works in Asterisk and it's important that we should only be focusing on the SDP.
We use SDP to be able to debug audio issues, codec issues and more for Zulu

**Deconstructing the client SDP lines**

**Global Lines**

| v=0 | protocol version number, currently only 0 |
| o-- 2276445737850684162 2 IN IP4 127.0.0.1 | The first number is the session id, an unique identifier for the session (2276445737850684162). The number in second position - 2 - is the session version: if a new offer/answer negotiation is needed during this media session this number will be increased by one. This will happen when any parameter need to be changed in the media session such as on-hold, codec-change, add-remove media track. The reasoning this is 2 at this point is because the first request was an unauthenticated INVITE to Asterisk which was rejected as "UNAUTHENTICATED". The three following fields are the network type (Internet), IP address type (version 4) and unicast address of the machine which created the SDP. These three values are not relevant for the negotiation. |
| s-- | The s line contains a textual session name, which is not commonly used as you can see here. |
| t=0 0 | It gives the starting and ending time. When they are both set to 0 like our case it means that the session is not bounded to a specific timing- in other words it's permanent and valid at any time. |
**a-group:** BUNDLE audio

**BUNDLE** groupings establishes a relationship between several media lines included in the SDP, commonly audio and video. In WebRTC it's use to multiplex several media flows in the same RTP session as described in draft-ietf-mmusic-sdp-bundle-negotiation. In this case the browser offers to multiplex both audio and video, but that has to be also supported and accepted by the other side.

**a=msid-semantic:** WMS

This lines gives an unique identifier for the WebRTC Media Stream (WMS) during the PeerConnection's life. This identifier that will be used in the a=msid attributes for each m-line belonging to a specific Media Stream (in our case both audio and video m-lines). This means that the RTP media stream (identified by the SSRC field present in every RTP packet) belongs to that media stream and that it is a track of that media stream. It is an explicit association of an individual RTP media stream to the MediaStream WebRTC object. For more info about this refer to draft-ietf-mmusic-msid

---

### Audio Lines

<table>
<thead>
<tr>
<th>m=audio 51083 UDP/TLS/RTP/SAVPF 111 103 104 9 0 8 106 105 13 110 112 113 126</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>m</strong> means it is a media line – it condenses a lot of information on the media attributes of the stream. In this order, it tells us:</td>
</tr>
<tr>
<td>- <em>audio</em> - the media type is going to be used for the session (media types are registered at the IANA),</td>
</tr>
<tr>
<td>- 51083 - the transport port to which the media stream is sent. The meaning of the transport port depends on the network being used as specified in the relevant &quot;c=&quot; field, and on the transport protocol defined in the transport protocol sub-field of the media field. Other ports used by the media application (such as the RTP Control Protocol (RTCP) port [19]) MAY be derived algorithmically from the base media port or MAY be specified in a separate attribute (for example, &quot;a=rtcp:&quot; as defined in [22]).</td>
</tr>
<tr>
<td>- UDP/TLS/RTP/SAVPF - the transport protocol is going to be used for the session, and last but not least</td>
</tr>
<tr>
<td>- 111 103 104 0 8 106 105 13 110 112 113 126 - the media format descriptions are supported by the browser to send and receive media.</td>
</tr>
<tr>
<td>RTP/SAVPF is defined in RFC5124. In short it requires the use of SRTP and SRTCP and RTCP Feedback packets.</td>
</tr>
<tr>
<td>The media format descriptions, with protocol RTP/SAVPF, gives the RTP payload numbers which are going to be used for the different formats. Payload numbers lower than 96 are mapped to encoding formats by the IANA. In our SDP <em>maps to G711U</em> and <em>maps to G711A</em>. Format numbers larger than 95 are dynamic and there are a=rtpmap: attributes to map from the RTP payload type numbers to media encoding names. There are also a=fmtp:attributes which specify format parameters</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>c=IN IP4 66.185.28.100</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>c</strong> is a connection line. This lines give the IP from where you expect to send and receive the real time traffic. As ICE is mandatory in WebRTC the IP in the c-line is not going to be used.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a=rtcp:64651 IN IP4 66.185.28.100</th>
</tr>
</thead>
<tbody>
<tr>
<td>This line explicitly specifies the IP and port that will used for RTCP, not derived from the base media port. Note that is the same port as for SRTP as RTCP multiplex is supported.</td>
</tr>
</tbody>
</table>

---

**ICE Candidates**
ICE is the protocol chosen for NAT traversal in WebRTC. You can find a very didactic and comprehensive explanation of ICE here. ICE is complex enough to deserve its own post, but I will try to explain its SDP lines in an understandable way.

Host candidate for RTP on UDP - in this ICE line our browser is giving its host candidates- the IP of the interface or interfaces the browser is listening on the computer. The browser can receive/send SRTP and SRTCP on that IP in case there is IP visibility with some candidate of the remote peer. For example, if the other computer is on the same LAN, hosts candidates will be used. The number after the protocol (udp) – 2122260223 - is the priority of the candidate. Notice that priority of host candidates is the higher than other candidates as using host candidates are more efficient in terms of use of resources. The first lines (component= 1) is for RTP and second line (component = 2) is for RTCP.

Host candidate for RTCP on UDP - these lines are the same as the two ICE lines before but for TCP traffic. Please note that the priority is the lower – i.e 1518280447 is smaller than 2122260222 - as TCP is not optimal for real-time media transportation.

Reflexive candidate for RTP over UDP - here we have the server reflexive candidates. Note that they have lower priority than host candidates. These candidates are discovered thanks to STUN server (see this
ICE lines are the mechanism chosen for NAT traversal in WebRTC. You can find a very didactic and comprehensive explanation of ICE here. ICE is complex enough to deserve its own post, but I will try to explain its SDP lines in an understandable way.

Reflexive candidate for RTCP over UDP - here we have the server reflexive candidates. Note that they have lower priority than host candidates. These candidates are discovered thanks to STUN server (see this).

ICE lines are the mechanism chosen for NAT traversal in WebRTC. You can find a very didactic and comprehensive explanation of ICE here. ICE is complex enough to deserve its own post, but I will try to explain its SDP lines in an understandable way.

Reflexive candidate for RTCP over UDP - here we have the server reflexive candidates. Note that they have lower priority than host candidates. These candidates are discovered thanks to STUN server (see this).

ICE is the protocol chosen for NAT traversal in WebRTC. You can find a very didactic and comprehensive explanation of ICE here. ICE is complex enough to deserve its own post, but I will try to explain its SDP lines in an understandable way.

Relay candidate for RTCP over UDP - next we have the relay candidates. Those candidates are obtained from a TURN server which must be provisioned when creating the peer connection. Note that the priority here is lower than the host and reflex candidates (25108222 is higher) so the relay will be used only if “hole punching” is not working with host and reflex candidates. Just for your curiosity we used the open source coturn project in this test (see our post on that here). The first IP/port couple corresponds to the IP and port assigned by the TURN server for this media session. raddr and rport correspond to the public IP and the public port through which the WebRTC endpoint is reaching Internet.

ICE Parameters

a=ice-ufrag:feeM
Once the ICE candidates are exchanged, a verification process starts where the Zulu client and Asterisk try to reach each other using the candidates provided. The ice-ufrag and ice-pwd credentials are used in that process to avoid receiving potential attacks from endpoints that are not involved in the session who could potentially create a media session without authorization.

a=ice-pwd:Mv4eWoFridPvphZr7sGSQdh1
Once the ICE candidates have been exchanged, a verification process starts where the Zulu client and Asterisk try to reach each other using the candidates provided. The ice-ufrag and ice-pwd credentials are used in that process to avoid receiving potential attacks from endpoints that are not involved in the session who could potentially create a media session without authorization.

a=ice-options:trickle
Identifies that this peer supports trickle ice. This might not be supported by the other peer however

DTLS Parameters

This fingerprint is the result of a hash function (using sha-256 in this case) of the certificates used in the DTLS-SRTP negotiation. This line creates a binding between the signaling (which is supposed to be trusted) and the certificates used in DTLS, if the fingerprint doesn’t match, then the session should be rejected.

a=setup:actpass
This parameter means that this peer can be the server or the client which starts the DTLS negotiation. This parameter was initially defined in RFC4145, which has been updated by RFC4572.

Audio Lines (Continued)
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=mid:audio</td>
<td>This is the identifier which is used in the BUNDLE line. In case we have different media we should have different identifiers for each.</td>
</tr>
<tr>
<td>a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level</td>
<td>RFC3550 defines the capability to extend the RTP header. This line defines extensions which will be used in RTP headers so that the receiver can decode it correctly and extract the metadata. In this case the browser is indicating that we are going to include information on the audio level in the RTP header as defined in RFC6464.</td>
</tr>
<tr>
<td>a=sendrecv</td>
<td>This line says that the browser is willing to both send and receive audio in this session. Other values could be sendonly, recvonly and inactive which are used to implement different scenarios like putting calls on-hold.</td>
</tr>
<tr>
<td>a=rtcp-mux</td>
<td>This line means that this peer supports multiplexing RTCP with RTP traffic.</td>
</tr>
<tr>
<td><strong>Codec Parameters</strong></td>
<td></td>
</tr>
<tr>
<td>a=rtpmap:111 opus/48000/2</td>
<td>Opus is one of the MTI audio codecs for WebRTC. It features a variable bit rate (6kbps-510kbps) and is not under any royalty so it can be freely implemented in any browser (unlike other codecs like as G.729). Opus support is starting to become common and it has become critical for most WebRTC applications.</td>
</tr>
<tr>
<td>a=rtcp-fb:111 transport-cc</td>
<td>Support for handling congestion. In a lossy network, these congestion control algorithms could tell the sender to lower the send bit rate. Lowering the send bitrate may reduce loss (at the expense of quality). However, if the network is always 10% lossy, like a noisy WiFi network, you still might suffer from video frame decode problems.</td>
</tr>
<tr>
<td>a=fmtp:111 minptime=10;useinbandfec=1</td>
<td>This line includes optional payload-format-specific parameters supported by Chrome for audio Opus codec. minptime=10 specifies the lowest value of the packetization time (ptime: the number of milliseconds of audio transported by a single packet). useinbandfec=1 specifies that the decoder has the capability to take advantage of the Opus in-band FEC (Forward Error Connection). For more info check RFC7587</td>
</tr>
<tr>
<td>a=rtpmap:103 ISAC/16000</td>
<td>The ISAC (Internet Speech Audio Codec) is a wideband speech codec for high quality conferences. The 16000 indicates that ISAC is going to be used at 16kbps. Because this is first, 16kbps will be considered before ISAC at 32kbps as specified in the next line.</td>
</tr>
<tr>
<td>a=rtpmap:104 ISAC/32000</td>
<td>The ISAC (Internet Speech Audio Codec) is a wideband speech codec for high quality conferences. The 32000 indicates that ISAC is going to be used at 32kbps. 16kbps will be considered before ISAC at 32kbps as specified in the previous line because it was first.</td>
</tr>
<tr>
<td>a=rtpmap:9 G722/8000</td>
<td>G722 is a wideband audio codec operating at 48, 56 and 64 kbit/s which provides improved speech quality due to a wider speech bandwidth of 50–7000 Hz compared to narrowband speech coders like G.711.</td>
</tr>
<tr>
<td>a=rtpmap:0 PCMU/8000 a=rtpmap:8 PCMA/8000</td>
<td>These lines specify G711 mu and a-law, which is a classic telecom 64kpbs pulse code modulation (PCM) codec using different companding laws. 0 and 8 are the static payload types for PCMU and PCMA respectively. Technically these lines do not to be present since this information can be inferred by the codec list in the media line - PCMU or PCMA.</td>
</tr>
<tr>
<td>a=rtpmap:106 CN/32000</td>
<td>The dynamic RTP payload types above (except payload type 13 which is static) indicate that Comfort Noise (CN) is going to be used for codecs of rates 48000, 32000, 16000 and 8000 kbits/s.</td>
</tr>
<tr>
<td>a=rtpmap:105 CN/16000</td>
<td>The dynamic RTP payload types above (except payload type 13 which is static) indicate that Comfort Noise (CN) is going to be used for codecs of rates 48000, 32000, 16000 and 8000 kbits/s.</td>
</tr>
</tbody>
</table>
The dynamic RTP payload types above (except payload type 13 which is static) indicate that Comfort Noise (CN) is going to be used for codecs of rates 48000, 32000, 16000 and 8000 kbits/s.

This line indicates the browser supports RFC4733, allowing it to send DTMFs within the RTP not as the usual digitized sine waves but as a special payload (in this case with payload 126 in the RTP packet). This DTMF mechanism ensure that DTMFs will be transmitted independently of the audio codec and the signaling protocol.

**SSRC Parameters**

The `cname` source attribute associates a media source with its Canonical End-Point Identifier which will remain constant for the RTP media stream even when the ssrc identifier changes if a conflict is found. This is the value that the media sender will place in its RTCP SDES packets.

This line is used to signal the association between the RTP concept of SSRC and the WebRTC concept of “media stream” / “media stream track” using SDP signaling [draft-ietf-mmusic-msid]. The first parameter corresponds to the id of the Media Stream and the second one the id of the Media Stream Track. These ids are handled in the WebRTC API. The first number is the SSRC identifier that will be included in the SSRC field of the RTP packets.

The `msid` parameter is deprecated and `mslabel` is kept for backward compatibility.

The `label` attribute is also deprecated by `msid` and carries a pointer to a RTP media stream in the context of an arbitrary network application that uses SDP. This label corresponds with the Media Stream Track id in the WebRTC API, which is included in the `msid` line.

**Asterisk**

You can see SDP in Asterisk by going into the CLI and running:

```
Apex-Sangoma*CLI> pjsip set logger on
PJSIP Logging enabled
```

Alternatively you can set the logger per host (which is a much better way to debug than getting everything)

```
Apex-Sangoma*CLI> pjsip set logger host 127.0.0.1
PJSIP Logging Enabled for host: 127.0.0.1
```

We can identify packets to be SDP because "Content-Type" is "application/sdp"

An example of a packet on Asterisk with SDP from pjsip set logger looks like this:
The important part here is the SDP itself, you'll notice if you don't focus specifically on just the SDP part that asterisk labels Via with "invalid". This is all part of how WebRTC works in Asterisk and it's important that we should only be focusing on the SDP
Deconstructing Asterisk's SDP lines

### Global Lines

<table>
<thead>
<tr>
<th>v=0</th>
<th>protocol version number, currently only 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>o=- 2085624096 4 IN IP4 192.168.1.4</td>
<td>The first number is the session id, an unique identifier for the session (2085624096). The number in second position - 4 - is the session version: if a new offer/answer negotiation is needed during this media session this number will be increased by one. This will happen when any parameter need to be changed in the media session such as on-hold, codec-change, add-remove media track. The reasoning this is 4 at this point is because the first request was an unauthenticated INVITE to Asterisk which was rejected as &quot;UNAUTHENTICATED&quot;. The three following fields are the network type (Internet), IP address type (version 4) and unicast address of the machine which created the SDP. These three values are not relevant for the negotiation.</td>
</tr>
<tr>
<td>s=Asterisk</td>
<td>The s line contains a textual session name</td>
</tr>
<tr>
<td>c=IN IP4 192.168.1.4</td>
<td>c is a connection line. This line gives the IP from where you expect to send and receive the real time traffic. As ICE is mandatory in WebRTC the IP in the c-line is not going to be used.</td>
</tr>
<tr>
<td>t=0 0</td>
<td>It gives the starting and ending time. When they are both set to 0 like our case it means that the session is not bounded to a specific timing- in other words it's permanent and valid at any time.</td>
</tr>
</tbody>
</table>

### Audio Lines

- `m=audio 12512 UDP/TLS/RTP/SAVPF 0 8 126`: This indicates that the media stream is for audio and is conveyed via UDP/TLS/RTP/SAVPF. The RTP port is 12512, and the SAVPF parameters are 0, 8, and 126.
- `a=connection:new`: Indicates that the media session is new.
- `a=setup:active`: Indicates that the media session is active.
- `a=ice-ufrag:4413544c44282ec57b2d9aef7fd62270`: This is the user password for ICE.
- `a=ice-pwd:319774777acc49493e676e0903ef5ba`: This is the password for ICE.
- `a=candidate:H534735e4 1 UDP 2130706431 fe80::290:27ff:feed:b7cc 12512 typ host`: An ICE candidate for the host.
- `a=candidate:Hc0a80104 1 UDP 2130706431 192.168.1.4 12512 typ host`: Another ICE candidate for the host.
- `a=rtpmap:0 PCMU/8000`: This line indicates the RTP format for PCMU codec.
- `a=rtpmap:8 PCMA/8000`: This line indicates the RTP format for PCMA codec.
- `a=rtpmap:126 telephone-event/8000`: This line indicates the RTP format for the telephone-event codec.
- `a=fmtp:126 0-16`: This line indicates the format specific options for the telephone-event codec.
- `a=ptime:20`: This line indicates the packet delay for the RTP stream.
- `a=maxptime:150`: This line indicates the maximum packet delay for the RTP stream.
- `a=sendrecv`: This line indicates that the media stream is both send and receive.
- `a=rtcp-mu`: This line indicates that the RTCP is mandatory.
m=audio 12512 UDP/TLS/RTP/SAVPF 0 8 126

\textit{m} means it is a media line – it condenses a lot of information on the media attributes of the stream. In this order, it tells us:

- \textit{audio} - the media type is going to be used for the session (media types are registered at the IANA),
- \textit{12512} - the transport port to which the media stream is sent. The meaning of the transport port depends on the network being used as specified in the relevant "c=" field, and on the transport protocol defined in the transport protocol sub-field of the media field. Other ports used by the media application (such as the RTP Control Protocol (RTCP) port [19]) MAY be derived algorithmically from the base media port or MAY be specified in a separate attribute (for example, "a=rtpmap:" as defined in [22]),
- \textit{UDP/TLS/RTP/SAVPF} - the transport protocol is going to be used for the session, and last but not least
- \textit{0 8 126} - the media format descriptions are supported by Asterisk to send and receive media.

RTP/SAVPF is defined in RFC5124. In short it requires the use of SRTP and SRTCP and RTCP Feedback packets.

The media format descriptions, with protocol RTP/SAVPF, gives the RTP payload numbers which are going to be used for the different formats. Payload numbers lower than 96 are mapped to encoding formats by the IANA. In our SDP \textit{a} maps to G711U and \textit{8} to G711A. Format numbers larger than 95 are dynamic and there are \textit{a=rtpmap:} attribute to map from the RTP payload type numbers to media encoding names. There are also \textit{a=fmtp:} attributes which specify format parameters

**DTLS Parameters**

\textit{a=connection:new}

The preceding description of the 'setup' attribute is placed in the context of using SDP to initiate a session. Still, SDP may be exchanged between endpoints at various stages of a session to accomplish tasks such as terminating a session, redirecting media to a new endpoint, or renegotiating the media parameters for a session. After the initial session has been established, it may be ambiguous whether a subsequent SDP exchange represents a confirmation that the endpoint is to continue using the current TCP connection unchanged, or is a request to make a new TCP connection. The media-level 'connection' attribute, which is charset-independent, is used to disambiguate these two scenarios. Valid values are "new" or "existing".

\textit{a=setup:active}

This parameter means that Asterisk starts is the DTLS client. This parameter was initialing defined in RFC4145, which has been updated by RFC4572.


This fingerprint is the result of a hash function (using sha-256 in this case) of the certificates used in the DTLS-SRTP negotiation. This line creates a binding between the signaling (which is supposed to be trusted) and the certificates used in DTLS, if the fingerprint doesn' t match, then the session should be rejected.

**ICE Parameters**

\textit{a=ice-ufrag:4413544c44282ec57b2d9aef7fd62270}

Once the ICE candidates are exchanged, a verification process starts where the Zulu client and Asterisk try to reach each other using the candidates provided. The \textit{ice-ufrag} and \textit{ice-pwd} credentials are used in that process to avoid receiving potential attacks from endpoints that are not involved in the session who could potentially create a media session without authorization.

\textit{a=ice-pwd:319774777acc49493e676e09093ef5ba}

Once the ICE candidates have been exchanged, a verification process starts where the Zulu client and Asterisk try to reach each other using the candidates provided. The \textit{ice-ufrag} and \textit{ice-pwd} credentials are used in that process to avoid receiving potential attacks from endpoints that are not involved in the session who could potentially create a media session without authorization.
**ICE Candidates**

ICE is the protocol chosen for NAT traversal in WebRTC. You can find a very didactic and comprehensive explanation of ICE here. ICE is complex enough to deserve its own post, but I will try to explain its SDP lines in an understandable way.

Host candidate for RTP on UDP - In this ICE line our browser is giving its host candidates - the IP of the interface or interfaces the browser is listening on the computer. The browser can receive/send SRTP and SRTCP on that IP in case there is IP visibility with some candidate of the remote peer. For example, if the other computer is on the same LAN, hosts candidates will be used. The number after the protocol (udp) - 2130706431 - is the priority of the candidate. Notice that priority of host candidates is the higher than other candidates as using host candidates are more efficient in terms of use of resources. The first lines (component = 1) is for RTP and second line (component = 2) is for RTCP.

**Codec Parameters**

These lines specify G711 mu and a-law, which is a classic telecom 64kbps pulse code modulation (PCM) codec using different companding laws. 0 and 8 are the static payload types for PCMU and PCMA respectively. Technically these lines do not to be present since this information can be inferred by the codec list in the media line - PCMU or PCMA.

This line indicates the browser supports RFC4733, allowing it to send DTMFs within the RTP not as the usual digitized sine waves but as a special payload (in this case with payload 126 in the RTP packet). This DTMF mechanism ensure that DTMFs will be transmitted independently of the audio codec and the signaling protocol.

This line includes optional payload-format-specific parameters supported by Asterisk for DTMF.

**Audio Lines (Continued)**

The number of miliseconds of audio transported by a single packet.

Maxptime specify the maximum amount of media that can be encapsulated in each packet, expressed as time in milliseconds. The size of the packet can have side effects in the quality of the audio and the BW. It is possible to modify this values in the SDP.

This line says that the browser is willing to both send and receive audio in this session. Other values could be sendonly, recvonly and inactive which are used to implement different scenarios like putting calls on-hold.

This lines means that this peer supports multiplexing RTCP with RTP traffic.

**Putting it all Together**

At this point you can use something advanced like Wireshark but you can probably deal with 99% of the issues you'll run into with Asterisk debugging and Zulu client debugging.

**Debugging No/One Way Calls**

**Example #1: Initiate a Call From Zulu**

Initiating a call from Zulu to *43 results in no audio on either end
Zulu Client

In the Zulu client when a call is initiated during setup you’ll notice the following

**RTCIceGatheringState changed: gathering**

Followed shortly thereafter by:

```
ICE candidate received: candidate:4033732497 1 udp 2122260223
192.168.0.104 55058 typ host generation 0 ufrag JAVR network-id 1
ICE candidate received: candidate:2066175263 1 udp 2122192127 2002:8ae5:
8e88:1::103 55059 typ host generation 0 ufrag JAVR network-id 2
ICE candidate received: candidate:2599619110 1 udp 2122129151 10.224.50.16
63369 typ host generation 0 ufrag JAVR network-id 3 network-cost 50
ICE candidate received: candidate:4033732497 2 udp 2122260222
192.168.0.104 62365 typ host generation 0 ufrag JAVR network-id 1
ICE candidate received: candidate:2066175263 2 udp 2122192126 2002:8ae5:
8e88:1::103 62366 typ host generation 0 ufrag JAVR network-id 2
ICE candidate received: candidate:2599619110 2 udp 2122129150 10.224.50.16
49869 typ host generation 0 ufrag JAVR network-id 3 network-cost 50
ICE candidate received: candidate:3203277665 1 tcp 1518280447
192.168.0.104 9 typ host tcptype active generation 0 ufrag JAVR network-id 1
ICE candidate received: candidate:899983855 1 tcp 1518212351 2002:8ae5:
8e88:1::103 9 typ host tcptype active generation 0 ufrag JAVR network-id 2
ICE candidate received: candidate:3564173014 1 tcp 1518149375 10.224.50.16
9 typ host tcptype active generation 0 ufrag JAVR network-id 3 network-cost 50
ICE candidate received: candidate:3203277665 2 tcp 1518280446
192.168.0.104 9 typ host tcptype active generation 0 ufrag JAVR network-id 1
ICE candidate received: candidate:899983855 2 tcp 1518212350 2002:8ae5:
8e88:1::103 9 typ host tcptype active generation 0 ufrag JAVR network-id 2
ICE candidate received: candidate:3564173014 2 tcp 1518149374 10.224.50.16
9 typ host tcptype active generation 0 ufrag JAVR network-id 3 network-cost 50
ICE candidate received: candidate:1826845682 2 udp 1685921534
66.185.28.100 49869 typ srflx raddr 10.224.50.16 rport 49869 generation 0
uffrag JAVR network-id 3 network-cost 50
ICE candidate received: candidate:1826845682 1 udp 1685921535
66.185.28.100 63369 typ srflx raddr 10.224.50.16 rport 63369 generation 0
uffrag JAVR network-id 3 network-cost 50
```

This is the local client determining and retrieving ICE candidates, the result of this process becomes your candidate list. The list is determined by using the listed STUN server provided to the Zulu client. In a few moments you will see that list being sent to Asterisk so that Asterisk can determine what candidates it wants to use. This is a list of all the ways the client has determined are routable for media to traverse through on my local network. Asterisk will do the same from it’s end

If you’ve noticed a delay while dialing a number from a Zulu client or when answer the delay is a result of the client determining the ICE candidate list to send back to Asterisk
An INVITE request is then sent to Asterisk with the SDP which has the list of candidates received above. This also contains all of the codecs and settings the client supports:

```plaintext
Thu Jun 14 2018 17:47:55 GMT-0700 (Pacific Daylight Time) | sip.transport |

sending WebSocket message:

INVITE sip:*43@myhost.com SIP/2.0
Via: SIP/2.0/WSS fo93pieho9q1.invalid;branch=z9hG4bK7812875
Max-Forwards: 70
To: <sip:*43@myhost.com>
From: <sip:904010@myhost.com>;tag=b0r948n56d
Call-ID: q1urm4d86pv7irde447p
CSeq: 7768 INVITE
Authorization: Digest algorithm=MD5, username="904010", realm="asterisk",
nonce="1529023675/0deca1542e81456a5557672ba6615984", uri="sip:*43@myhost.com",
response="4d9c39fe7744e4a8cf29ef6e8795d034", opaque="28cddce66897ddcf", qop=auth,
cnonce="d4n4gpcpch8k", nc=00000001
Contact: <sip:iln2odb2@fo93pieho9q1.invalid;transport=wss;ob>
Allow: ACK,CANCEL,INVITE,MESSAGE,BYE,OPTIONS,INFO,NOTIFY,REFER
Supported: outbound
User-Agent: Zulu
Content-Type: application/sdp
Content-Length: 2867

v=0
o=- 2276445737850684162 2 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE audio
a=msid-semantic: WMS km0kdiSIfNWE5sIKmB1V6vfJZnQtM6kfOhJh
m=audio 51083 UDP/TLS/RTP/SAVPF 111 103 104 9 0 8 106 105 13 110 112 113 126
 c=IN IP4 66.185.28.100
 a=rtcp:64651 IN IP4 66.185.28.100
 a=candidate:4033732497 1 udp 2122260222 192.168.0.104 54930 typ host
generation 0 network-id 1
 a=candidate:2066175263 1 udp 2122192127 2002:8ae5:8e88:1::103 54931 typ
 host generation 0 network-id 2
 a=candidate:2599619110 1 udp 2122129151 10.224.50.16 51083 typ host
generation 0 network-id 3 network-cost 50
 a=candidate:4033732497 2 udp 2122260222 192.168.0.104 51360 typ host
generation 0 network-id 1
 a=candidate:2066175263 2 udp 2122192126 2002:8ae5:8e88:1::103 51361 typ
 host generation 0 network-id 2
 a=candidate:2599619110 2 udp 2122129150 10.224.50.16 64651 typ host
generation 0 network-id 3 network-cost 50
 a=candidate:3203277665 1 tcp 1518280447 192.168.0.104 9 typ host tcptype
 active generation 0 network-id 1
 a=candidate:899983855 1 tcp 1518212351 2002:8ae5:8e88:1::103 9 typ host
tcptype active generation 0 network-id 2
 a=candidate:3564173014 1 tcp 1518149375 10.224.50.16 9 typ host tcptype
 active generation 0 network-id 3 network-cost 50
 a=candidate:3203277665 2 tcp 1518280446 192.168.0.104 9 typ host tcptype
```
After the client has sent the invite (you will see this in Asterisk as well). Asterisk will send back a "TRYING" response.
<--- Transmitting SIP response (334 bytes) to WSS:127.0.0.1:60890 --->
SIP/2.0 100 Trying
Via: SIP/2.0/WSS fo93pieho9ql.invalid;rport=60890;received=127.0.0.1;
branch=z9hG4bK2467288
Call-ID: q1urmi3sjuf4a92pj3ks
From: <sip:904010@myhost.com>;tag=c9fug6liep
To: <sip:*43@myhost.com>
CSeq: 9345 INVITE
Server: PBXact-13.0.195.4(13.18.3)
Content-Length: 0

Followed by the SDP for Asterisk
What went wrong?

Looking at the SDP from Asterisk we can see there are only two candidates from Asterisk "fe80::290:27ff:feed:b7cc" and "192.168.1.4" these are being sent back to the Zulu client to use for audio routing. We know this is not route-able by our client because we are not on the same network. Since this is SDP from Asterisk we will have to configure Asterisk to work with us.

At this point you should try to add a STUN server into Asterisk SIP Settings under "RTP Settings". This might not always work, especially if the network is restricted and doesn't allow UDP packets out of it's firewall.

Setting a STUN Server
### RTP Settings

<table>
<thead>
<tr>
<th>Feature</th>
<th>Setting 1</th>
<th>Setting 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTP Port Ranges</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start:</td>
<td>10000</td>
<td></td>
</tr>
<tr>
<td>End:</td>
<td>20000</td>
<td></td>
</tr>
<tr>
<td>RTP Checksums</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Strict RTP</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>STUN Server Address</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>stun.l.google.com:19302</td>
<td></td>
</tr>
</tbody>
</table>

After a reload we try to dial out again

⚠️ You are experiencing connectivity issues.

**FT**

**Featurecode: echotest (infoser...**

*43*
Unfortunately setting a STUN server didn’t work in this case.

**Setting ICE Blacklist**

Subnets to exclude from ICE host, srflx and relay discovery. This is useful to optimize the ICE process where a system has multiple host address ranges and/or physical interfaces and certain of them are not expected to be used for RTP. For example, VPNs and local interconnections may not be suitable or necessary for ICE. Multiple subnets may be listed. If left unconfigured, all discovered host addresses are used.

The format for these overrides is: [address] / [subnet]

This is most commonly used for WebRTC
Next up let's try blacklisting candidates. Blacklisting does exactly what the description says. It will completely remove candidates from the list of ICE candidates Asterisk sends back to Zulu.

Not that this probably won't work since we only have two candidates in the list and one is IPv6 which we can't use anyways.

Head back over to Asterisk SIP Settings and let's see if it's any better if we add 192.168.1.0/24 here

```
<table>
<thead>
<tr>
<th>IP Addresses</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.168.1.0</td>
</tr>
<tr>
<td>192.168.1.24</td>
</tr>
</tbody>
</table>
```

Save and reload.

Then dial out from Zulu.

---

You are experiencing connectivity issues.

* FT

Featurecode: echotest (infoser...)

*43
We have definitely removed "192.168.1.4" from our list of candidates (remember the C and O lines are not used for routing media). However now we only have an IPv6 address which we can't use!

Remove the settings from ICE Blacklist

Add ICE Host Candidates Maps

When Asterisk is behind a static one-to-one NAT and ICE is in use, ICE will expose the server's internal IP address as one of the host candidates. Although using STUN (see the 'stunaddr' configuration option) will provide a publicly accessible IP, the internal IP will still be sent to the remote peer. To help hide the topology of your internal network, you can override the host candidates that Asterisk will send to the remote peer.
IMPORTANT: Only use this functionality when your Asterisk server is behind a one-to-one NAT and you know what you’re doing. If you do define anything here, you almost certainly will NOT want to specify 'stunaddr' or 'turnaddr' above.

The format for these overrides is: [local address] => [advertised address]

This is most commonly used for WebRTC

Next up let's try adding ICE Host candidate mapping

Here we are going to map our internal server address of "192.168.1.4" to our known external IP address of "174.71.YYY.XXX"

<table>
<thead>
<tr>
<th>ICE Host Candidates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Candidates</td>
</tr>
<tr>
<td>192.168.1.4 =&gt; 174.71.YYY.XXX</td>
</tr>
</tbody>
</table>

Save and reload

Then dial out from Zulu

Featurecode: echotest (infoser...
*43
↑ 00:39

<--- Transmitting SIP response (1343 bytes) to WSS:127.0.0.1:37740 --->
SIP/2.0 200 OK
Via: SIP/2.0/WSS 38udi0q1edo.invalid;rport=37740;received=127.0.0.1;
branch=z9hG4bK4826084
Call-ID: a7huja6kfhfr72hi2s
From: <sip:904010838880154.deployments.pbxact.com>;tag=rh231qepqd
To: <sip:*43@38880154.deployments.pbxact.com>;tag=4071d31a-a8d3-4244-91f5-bca4d1687b26
CSeq: 6574 INVITE
Server: PBXact-13.0.195.4(13.18.3)
Contact: <sip:127.0.0.1:37740;transport=ws>
Allow: OPTIONS, SUBSCRIBE, NOTIFY, PUBLISH, INVITE, ACK, BYE, CANCEL,
UPDATE, PRACK, REGISTER, REFER, MESSAGE
Supported: 100rel, replaces, norefersub
P-Asserted-Identity: "Echo Test" <sip:*43@38880154.deployments.pbxact.com>
Content-Type: application/sdp
Content-Length: 665

v=0
o=- 2867060712 4 IN IP4 192.168.1.4
s=Asterisk
Success! We now have audio in both directions!

We can now see that the invalid candidate of

```
  a=candidate:Hc0a80104 1 UDP 2130706431 192.168.1.4 11716 typ host
```

has changed to

```
  a=candidate:Hae479ced 1 UDP 2130706431 174.71.YYY.XXX 16082 typ host
```

Tools/Tests

- https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/
- https://test.webrtc.org/

Reference

- https://temasys.io/webrtc-ice-sorcery/
- https://www.slideshare.net/saghul/ice-4414037 (old 2010)
- https://testrtc.com/webrtc-api-trace/
- https://webrtcchacks.com/sdp-anatomy/
- https://medium.com/xirsys/webrtc-connectivity-woes-and-you-dffb59f0d582