WebSOCKET based Real time text (RTT)

WebRTC gateway
For WebRTC and SIP interop

Version 2.5a

Authors:

Emmanuel Buu, Ivés. emmanuel.buu@ives.fr www.ives.fr
Gunnar Hellström, Omnitor. Gunnar.hellstrom@omnitor.se www.omnitor.se

Version history:

Date	Version	Authors
	1.0	Emmanuel BUU, Ives
	1.1	Emmanuel BUU, Ives – degraded network conditions
	2	Emmanuel BUU, Ives, Added section 3 - interoperability
June 18, 2014	2.1	Gunnar Hellström, Omnitor - Precision on text negotiation and fast update request handling
June 19, 2014	2.2	Gunnar Hellström, Omnitor- language adjustments
June 28, 2014	2.3	Gunnar Hellström, Omnitor - change from automatic date to manual date
July 4, 2014	2.4	Gunnal Hellström, Omnitor - security enhancements
June 30, 2015	2.5	Gunnar Hellström, Omnitor - credits

Contents

1 Overview

IVeS is building a WebRTC gateway as part of a project with Omnitor. One of the goal of this project is to provide Total Conversation using a WebRTC client and enable to interoperate with regular SIP based Total Conversation clients.

Total Conversation as implemented in the regular SIP world enable the use of any combination of audio, video and Real-Time Text (RTT) media. In regular SIP, RTP is used to carry all media. Real-time text is based on T.140 packetized over RTP using RFC 4103. RFC 2198 redundancy is used to achieve reliability.

WebRTC is an HTML5 standard that enables real-time communication using standard browser features. The supported media are audio and video. The current implementation of WebRTC uses audio and video codecs that are not generally supported in the sip world: VP8 for video and OPUS for audio.

The WebRTC standard provides a data exchange mechanism called "data channel" that is not yet ready for production, especially regarding NAT traversal issues.

1.1 Purpose of this document

The present document proposes a way to carry real-time text over Websocket and negotiate it over SIP in an interoperable manner. The mechanism is implemented in a system for WebRTC based RTT and Total Conversation, with interoperability with traditional SIP.

The system is implemented by Ivés and Omnitor in a project called RERC-Telecommunications Access.

The specification is made so that it shall be simple to replace the Websocket mechanism for RTT with a WebRTC data channel based mechanism when the WebRTC data channel specifications and implementations are mature.

Section 2 proposes a real-time text transport mechanism from a general point of view.

Section 3 proposes an interoperability mechanism between WebRTC total conversation and SIP based total conversation.

Section 4 select implementation choices and limits for the current project.

1.2 Acknowledgement

The contents of this resource were developed in part with funding from the National Institute on Disability, Independent Living, and Rehabilitation Research (NIDILRR grants # H133E040013, & 90RE5003-01-00) in the Administration for Community Living (ACL), Department of Health and Human Services (HHS). The contents do not necessarily represent the policy nor endorsment of NIDILRR, ACL, or HHS.

2 WebRTC based total conversation

2.1 Signalling

In order to enhance interoperability with regular SIP client, some restructuring and practices are proposed for using WebRTC technology

> WebRTC based total conversation clients shall use the SIP protocol as signalling and comply with RFC 3261. Support for Websocket and Websocket over TLS based transports as specified by RFC 7118 is mandatory. UDP and TCP transport are not supported.

As Web browsers do not implement server mode, we mandate the use of a SIP server.

> WebRTC based total conversation clients must register on a SIP server. The registration process start by establishing a Websocket connection with the server and then continues by performing a SIP registration as specified by RFC 3261.
>
> WebRTC based total conversation clients must maintain the Websocket connection open permanently until the user decides to close it and unregister.
>
> The SIP server shall always reuse an open Websocket connection and never attempt to open a Websocket connection to the client.
>
> If the Websocket connection disconnects, the client shall attempt a new connection and refresh its registration by sending a new REGISTER message and increment Cseq field.

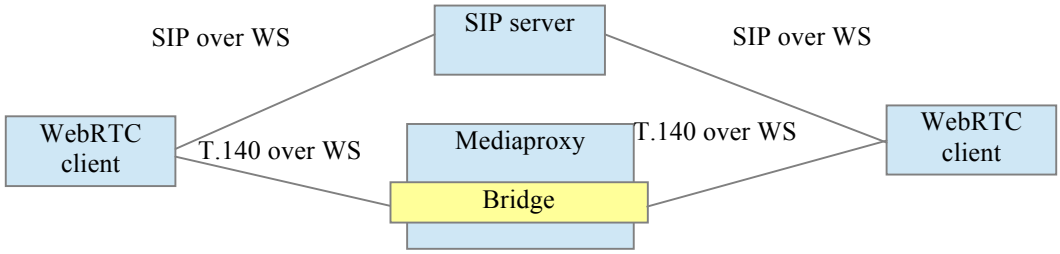
2.2 Real-time text over websocket

As WebRTC implementations does not support real-time text over RTP, we propose to use Websocket as transport for real-time text. Websocket over TLS can be used to secure cipher the text communications over the open Internet.

The total conversation clients shall negotiate real-time text as specified in the next section. If both client support real-time text, they will obtain an URI of a websocket connection to a **media proxy**.

Both clients shall then open a websocket connection to the media proxy. Real-time text is the exchanged over these connections as T.140 payload. A transmission interval of 300 ms is recommended.

No redundancy mechanism is necessary as Websocket is a reliable transport. However, network conditions may degrade and cause a websocket connection to disconnect or block.



In case of disconnect, the media proxy shall insert in the T.140 stream one Unicode Replacement Character to notify the other leg of the stream interruption.

If the websocket is blocked, the media proxy will fail to push the T.140 stream into it. In case the media proxy

fails or blocks sending T.140 data into this connection, it shall disconnect the websocket and insert a Replacement Character into the other websocket to indicate a possible loss of data.

In the case that the Websocket connection for real-time text is broken, the client shall make an effort to re-establish a Websocket connection using the same URL. If it cannot be deducted that text was not lost or duplicated because of the broken and reestablished connection, the Unicode Replacement Character shall be inserted by both the client and the server in the received text at the point where text may have been lost or duplicated because of the broken and reestablished connection. If reestablishment is not successful, the Replacement Character shall also be inserted as a marker for a possible character loss.

2.3 Offer / answer model

When establishing a call, a WebRTC based total conversation client shall comply with RFC 3264.

2.3.1 Negociating **REAL-TIME** text

A WebRTC based total conversation client shall add an additional media stream compliant with RFC 4145 to the offer, generated by the WebRTC API:

```
m=text 54321 TCP/WS t140
c=IN IP4 192.0.2.1
a=setup:active
a=connection:new
a=sendrecv
```

It indicates to the media proxy that the WebRTC client is ready to offer text over Websocket using T140 format and it will initiate the Websocket connection. The port and the IP address mentioned here are meaningless.

The SIP server shall interpret this media stream definition and contact one or several media proxies and prepare a text session. The communication between the SIP server and the media proxy is not part of this specification.

If the client requests secure Websocket, the offer should include TCP/WSS instead of TCP/WS.

When forwarding the message to the callee, the SIP server shall change the media description of the offer as follow:

```
m=text <port> TCP/WS t140
c=IN IP4 <mediaproxy address>
a=setup:passive
a=connection:new
a=sendrecv
a=ws://mediaproxy:port/rest/of/url
```

Where

<mediaproxy address> is the hostname or the IP address of the mediaproxy. It is ignored by the client.

<port> is the port the of the websocket server (ignored by the called client too)

The SIP server shall insert **one** SDP attribute « a=ws » (or a=wss, if secure WebSocket is requested) that is the URL of the Websocket connection to reach the media proxy.

When receiving this offer the called client shall open an websocket connection to the media proxy using the URL specified by the attribute « a=ws ». It shall answer as follow :

```
m=text 54321 TCP/WS t140
c=IN IP4 192.0.2.2
a=setup:active
a=connection:new
a=sendrecv
```

The SIP proxy shall generate a new Websocket URL and rewrite the answer as follow

```
m=text <port> TCP/WS t140
c=IN IP4 <mediaproxy address>
a=setup:passive
a=connection:new
a=sendrecv
a=ws://mediaproxy:port/rest/of/otherurl
```

When receiving this answer the caller client shall open its Websocket for real-time text. Once the call is established, both clients may exchange real-time text by sending and receiving T.140 data over their respective Websocket connection.
If the called client does not support real-time text or is not willing to use this media for the conversation, it should disable the media altogether by sending an « m » line with the port value set to zero in the SDP answer.

```
m=text 0 TCP/WS t140
```

2.3.2 Conversation with multiple text streams

In some situation (conferences, subtitle in several langages), it is desirable to establish several text stream in a single conversation.

To be completed.

2.3.3 Handling audio and video streams

WebRTC clients mandate the use of ICE and TURN for NAT traversal for audio and video streams. Several options can be implemented:

Option 1 : add a TURN server in the ICE candidates

When two WebRTC client call each other, the SIP server may add in the ICE candidate list of SDP offer and of the SDP answer, the address of a TURN server. This candidate shall have a low priority to encourage establishment of peer to peer media stream communications.

Option 2 : act as media proxy

In this case, the SIP server specifies one single media proxy as ICE candidate in offers and answers to force media to be processed by a media proxy.

2.4 Security

In order to enforce conversation confidentiality, it is recommended to use WebSocket over TLS for both SIP signalling and text media.

It is expected that the WebSocket URL contains a one time token valid during the call to avoid outsider to connect to the Websocket server and interfere with the conversation.

It is also expected that the media proxy accepts only one connection per URL.

3 Interoperability with regular SIP clients

One of the purposes of Total Conversation is to provide interoperability. This section extends the requirements described in section two in order to enable interoperability between WebRTC based total conversation clients and SIP based total conversation client. It also propose some call procedures to achieve call interoperability.

3.1 Principle of interoperability between WebRTC based and SIP based TC clients

Having selected a common signaling protocol for both type of client enables a natural interoperability at signaling level. But in order to achieve call interoperability, the media level needs to interoperate. The following issues needs to be addressed:

- There is often no common video codecs between SIP client and WebRTC client as current WebRTC implementations does not support H.264 and few SIP based total conversation clients support VP8.
- SRTP-DTLS is not supported by most SIP clients and we cannot expect a wide adoption soon.
- WebRTC mandate the support of ICE, RTCP feedback and a clever congestion control algorithm, RTCP multiplexing and so on. Interaction between a WebRTC client and a client that do not support these extension is unspecified.
- WebRTC favor a new audio codec called OPUS that is not widely supported. Falling back to G.711 is allowed but degrades the audio quality significantly.
- WebRTC real-time text as specified in section 2 does not interoperate naturally with real-time text on RTP transport as specified in RFC 4103.

The designers of WebRTC brought a lot of needed improvement on media transport but did not specify interoperability with legacy systems. In order to achieve this interoperability, several principles are possible:

- When a WebRTC and a legacy SIP client interoperates, the media proxy need to process the media and decapsulate it, and perform the necessary audio and video transcoding

- The SIP server should be aware of the nature of both client at each side of the conversation and may engage or not media proxy if both client use the same technology.

- The criteria used to discriminate a WebRTC client from a SIP client is the transport of the SIP protocol. Use of WebSocket means WebRTC client.

Technology/media feature	**WebRTC client**	**SIP client**
H.264 video codec	May support, will be offered.	Supported.
VP8 video codec	Supported	May support, will be offered.
OPUS audio codec	Supported	May support, will be offered
SRTP-DTLS	Supported	Not supported
RTCP Feedback	Supported	May support, will be offered
ICE	Supported	May support but will NOT be offered
Real-time text	Addition specified in section 2	May support RTT over RTP (RFC 4103), will be offered

Table showing the support of media and features in the two types of clients involved. “May support” means that the gateway supports the feature, and it depends on the selected client if the support is used.

This conversion and support requires a SIP server and a media proxy to comply with additional requirements listed in the next sections.

3.2 Interoperability requirements for the SIP server

The SIP server shall support SIP as described in RFC 3261, including UDP and TCP transport for SIP. The SIP server shall act as proxy and registrar as seen by the client[1].

3.3 Interoperability requirements for the Media Proxy

The Media proxy shall be able to handle SRTP-DTLS, act as an ICE candidate and decode the audio and video media streams from client.

The Media proxy shall be able to handle video stream transcoding from H.264 to VP8 and vice and versa.

The Media proxy shall support RTCP feedback, especially nack and fir as well as Audio Redundancy payload and Forward Error Correction used by some WebRTC implementations

The media proxy should support tmmbr and goog-remb messages, and implement congestion control algorithm as specified by http://ietfreport.isoc.org/idref/draft-alvestrand-rtcweb-congestion/ and known as « zero artefact »

The media proxy should be able to transcode from OPUS codec to G.722 or G.722.1 codec in order to preserve the high quality audio brought by WebRTC and interoperate with SIP client that

The media proxy shall be able to process real-time text both as specified in section 2 and as specified in RFC 4103. The media proxy shall be able to translate between the two modes.

3.4 Call procedures for acheiving interoperability

Three possible strategies can be implemented by the SIP server to provide interoperability with regular total conversation SIP client.

For clarity of explanation, let us assume that WebRTC client « Alice » calls SIP based client « Bob ».

3.4.1 Call procedure A : no prior client type detection

Step 1 - Alice calls. The INVITE packet comes to the SIP server. Because the contact contains a SIP uri with « transport=ws », the SIP server infers that « Alice » is using a WebRTC client.

The SDP contains : OPUS, G.711 codecs for audio, VP8 for video. T.140 over WebSocket as real-time text.

Step 2 – The SIP proxy contacts a media proxy

It modifies the INVITE packet as follow :

- It sets the media proxy address as the only ICE candidate.
- It adds H.264 as supported video codec
- It adds G.722 supported codec.
- If text media is requested by Alice, it adds a second text media with RTP using SAVPF support (see section …)
- It keeps all RTCP feedback and cryptology attributes.

---

1 Here the SIP server is a logical entity rather than a physical system. It may be composed of a separate proxy and registrar but it has to provide these two functions.

Step 3 – Bob's client receives the INVITE packet

As it is a well behaved client, it does not support « RTP/SAVPF » profile nor the « TCP/WS » profile and rejects the call with 488 Not Accepted Here as no media can be negotiated.

Step 4 – The SIP server process the 488 response and reissue a new INVITE packet (with increased Cseq) and a new SDP that uses media with « RTP/AVP » profile only. Crypto attribute and ICE attributes are removed. RTCP feedback attributes are kept.

The SIP server knows the « Bob » is a SIP client.

Step 5 - « Bob » receive the new INVITE and answer the call with 200 OK.

It select H.264 ang G.722 as audio and video codecs. It select T.140 with redundancy level 3 for real-time text.

Step 6 – SIP server processes the answer

It engages the transcoders and modify the SDP as follow

- It select VP8 as video codec
- It select OPUS as audio codec
- It answer to real-timetext over websocket.
- It adds ICE candidate for the mediaproxy
- it adds DTLS crypto attribute for the media proxy

The answer is then sent to Alice.

Step 7 - Alice receives the 200 OK answer

It processes it and establish the media streams with the media proxy.
It sends the ACK message that starts the transcoders
The ACK message is forwarded to Bob that start sending and receiving transcoded media from and to the media proxy.

The limitation of this procedure is :

- We cannot assume that all SIP clients will behave as described above even if the 488 answer is the most logical answer to expect in that case.
- It adds a message exchange and make the call connection a bit longer.

We also note that the step 4 of this procedure requires the SIP server to act as back to back user agent (B2BUA).

3.4.2 Variant of call procedure A

In order to overcome the first issue in the previous procedure, we may – in step 3, sends an INVITE with a « non-WebRTC » SDP. Then we would make sure that WebRTC clients answers 488 in this case.

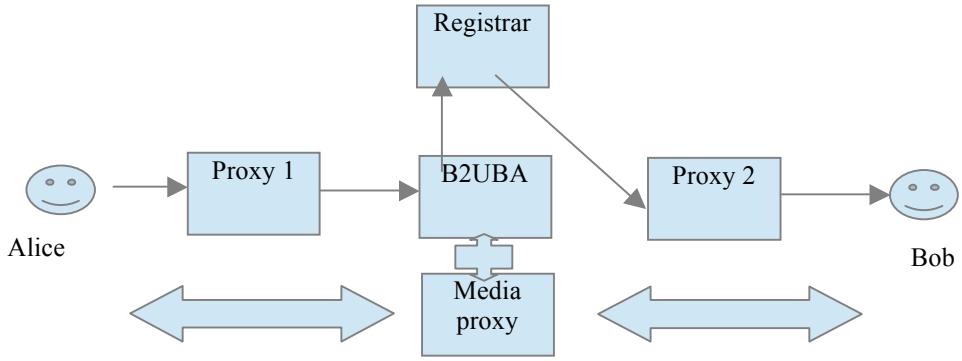
This is possible as this project provides its own WebRTC client. However it does not accommodate future WebRTC clients.

3.4.3 Call procedure B with client type detection

As the SIP server act as registrar, it stores the contact of each client and is therefore able to know which type of client it needs to reach. Even if we consider cases when the SIP server needs to forward calls to an ACD or another server, it can safely assume that those servers are « SIP clients ». In that case, the SIP server can directly create the « right » SDP suited for the client type.

The SIP server itself may not be a single physical entity.

Here is an example of distributed SIP infrastructure that may act as a SIP server.



In the diagram above, the subsystem that is aware of the nature of Bob is the registrar. The subsystem that is able to execute the core of the call procedure is the B2BUA. The call procedure needs to be augmented as follow:

When the B2BUA needs to identify the type of client in order to generate the appropriate SDP,

- It should query the registrar with an OPTIONS bob@registrar message.
  - The registrar may answer directly and specify the contacts of the client
  - The registrar may forward the OPTIONS message to the client and relay the answer to the B2BUA
  - In both cases, by examining the contact, the B2BUA will be able to decide which SDP to use.
- It may forward the call to the registrar. In that case, it is expected that the registrar replies « 488 not acceptable here » if the SDP of the call does not match the type of client.

3.5 Specific procedures for media interoperability

3.5.1 Support of RTCP FIR and RFC 5168

Some older SIP clients do not support RTCP Picture Loss Indicator or Fast Intraframe Request. Well behaved SIP that support those messages should indicate it in the SDP using « rtcp-fb » attributes.

When SIP server and the media proxy connect a WebRTC client Alice with a SIP client Bob that do not support RTCP FIR,

Case 1 : there are **no** video transcoder engaged

- Upon receiving either RTCP FIR, the media server shall process it and generate a new intraframe.
- Upon receving a Fast Update Request as specified in RFC 5168, the SIP server should contact the media proxy. The later shoud generate an intraframe.

Case 2 : there are video transcoders engaged

Media proxy and SIP server should

- Translate RTCP FIR into Fast Update Request as specified in RFC 5168
- Translate Fast Update Request as specified in RFC 5168 into RTCP FIR

3.5.2 Double offering for text media in call procedure A

In the step 2 of call procedure A, if text needs to be negotiated, the new SDP shall include two « text » media

in the offering. One with RTP/SAVPF transport, one with TCP/WS transport. This is needed as the SIP server does not know which kind of client is being called.

This is done by defining an SDP media group FID according to RFC 5888

```
a=group:FID textoverws textrfc4103
```

(…)

```
m=text <ws port> TCP/WS t140
c=IN IP4 <mediaproxy address>
a=setup:passive
a=connection:new
a=sendrecv
a=ws://mediaproxy:port/rest/of/otherurl
a=mid:textoverws
m=text <rtp port> SAVPF 102 101
a=fingerprint:....
a=setup:passive
a=connection:new
a=sendrecv
a=ws://mediaproxy:port/rest/of/otherurl
a=mid:textrfc4103
a=rtpmap:102 RED
a=rtpmap:101 T140

a=fmtp:102 101/101/101
```

A well behaved SIP client should select the media that it can process and reject the others by answering with an « m » line with port 0.

4 Implementation choices and uses cases

This chapter describes the implementation by Ivés and Omnitor of the principles specified in the previous chapters.

4.1 Actors

WebRTC client1

WebRTC client1 is made of Mobicent HTML5 WebRTC client (with some corrections), run on Chrome (add version).

WebRTC client2

WebRTC client2 is the same software except it is run on Firefox web brower.

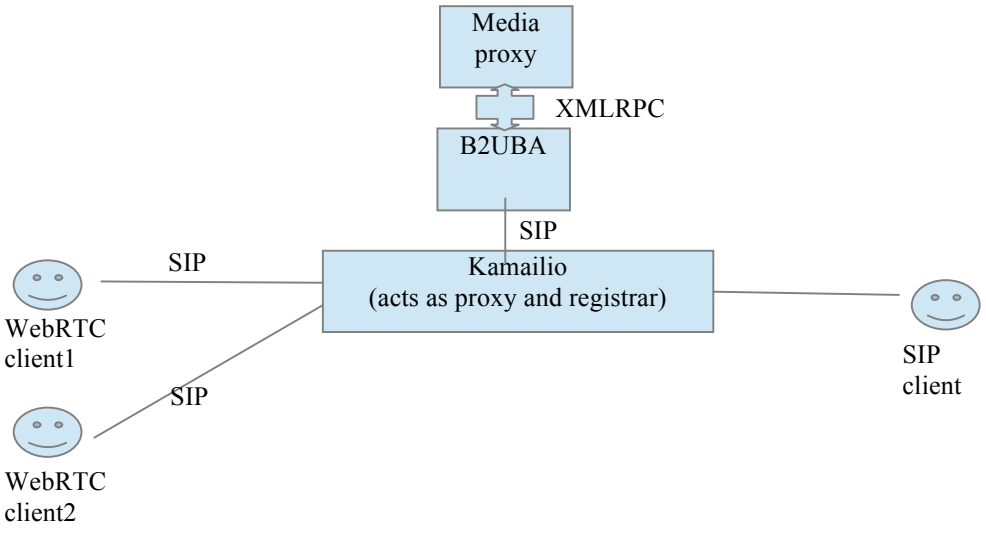
Regular SIP client

Regular SIP client with total conversation (Video, RTT and audio) support

SIP server

See next section

4.2 Details of sip server



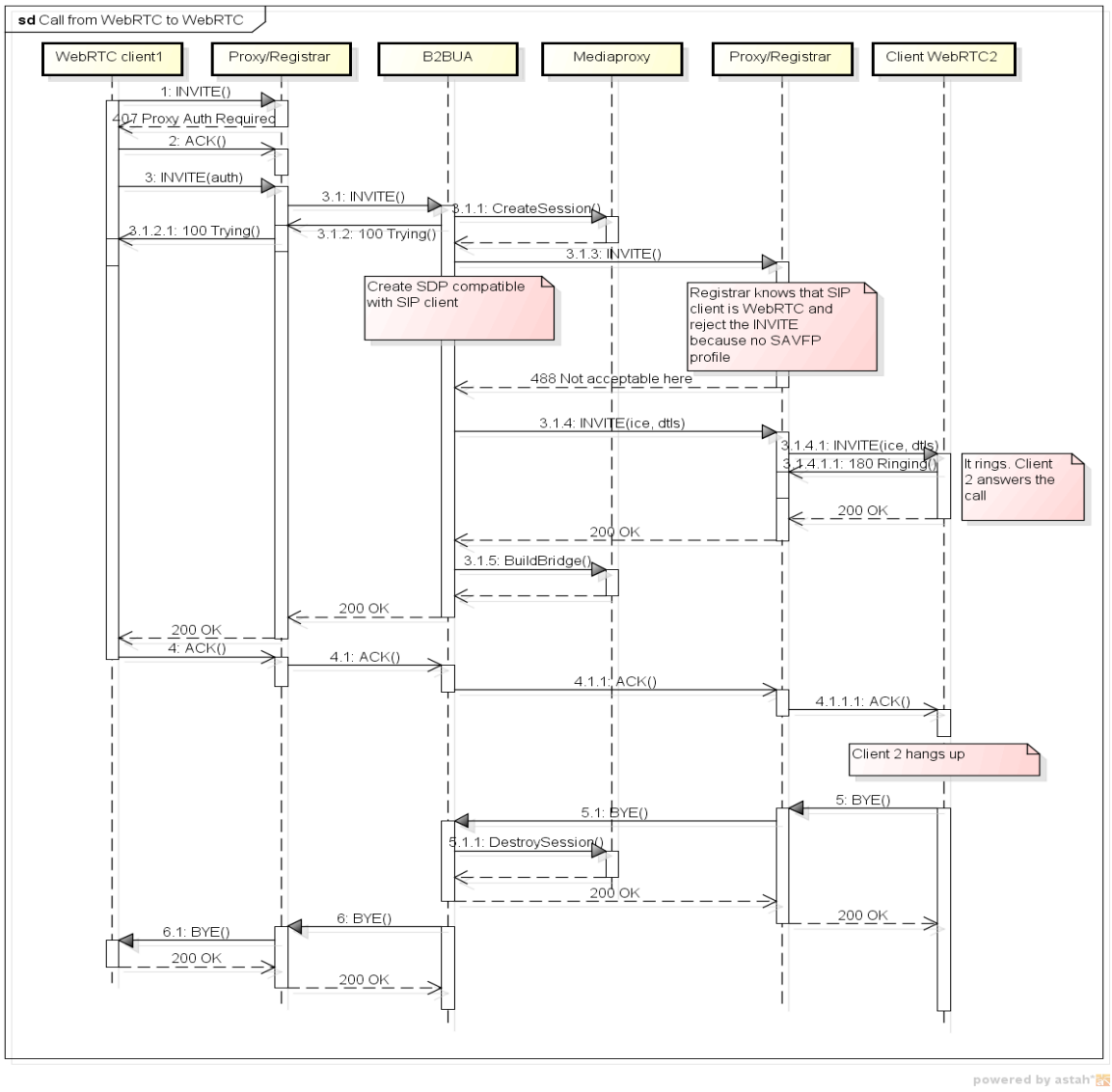
4.3 Features supported

Feature	
Audio codecs supported	G.711, Speex, OPUS, G.722
Video codec supported	H.263, H.264, VP8
RTCP Feedback	FIR, TNMBR, NACK, GOOGLE-REMB
Zero artefact support	Yes
DTLS-SRTP	Yes
Real-time text	T.140 over RTP (RFC4103) and T.140 over WS
Call procedure selected	Call procedure B

Limitations (that can be reduced later)

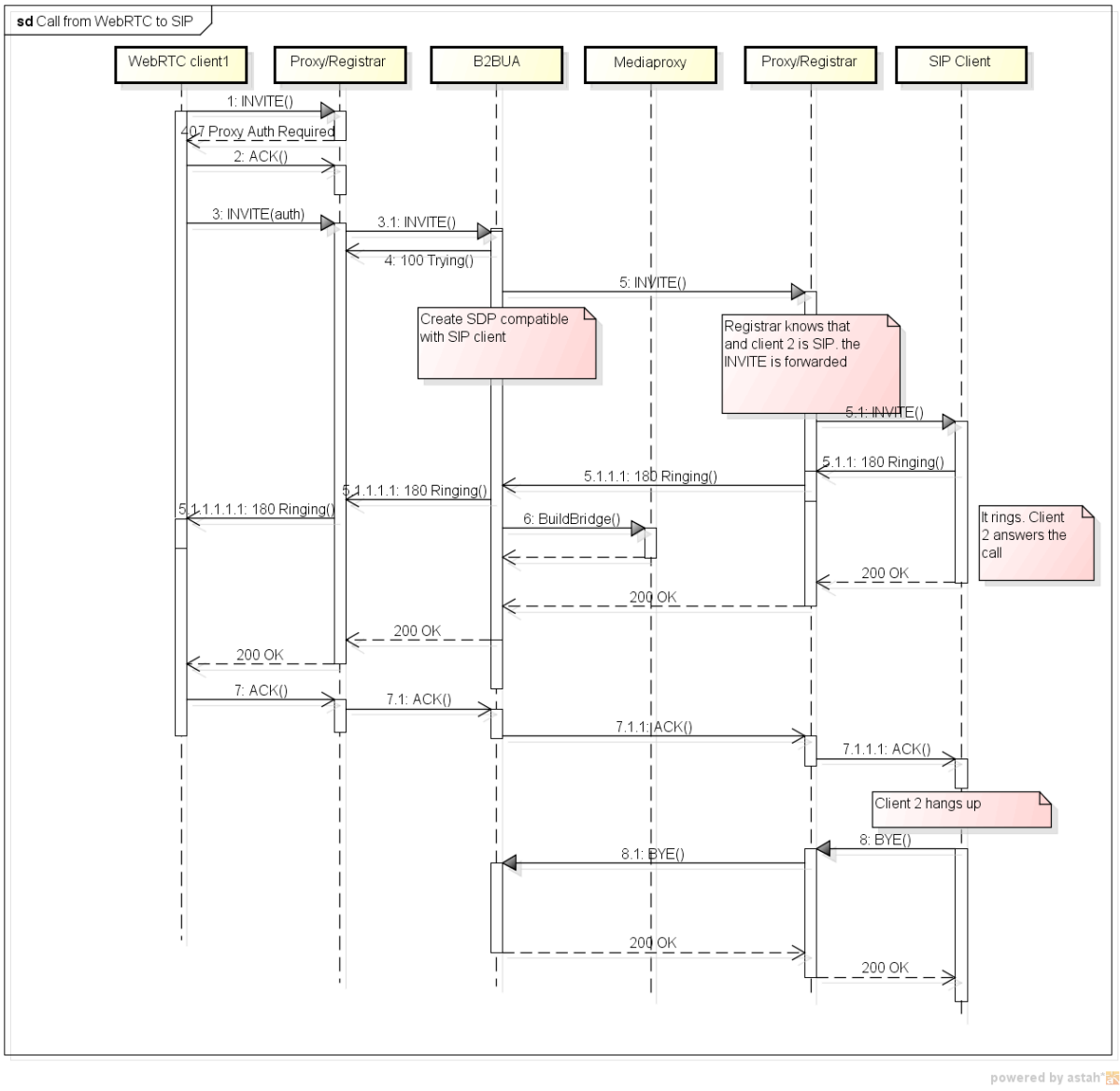
- Behavior in case of multiple registrations is undefined.
- Full implementation of re-INVITE / UPDATE will be gradual.
- First release of GW will not provide the ability to use peer to peer media.
- Gateway will not support media bundling as **draft-ietf-mmusic-sdp-bundle-negotiation-07**

4.4 Use cases

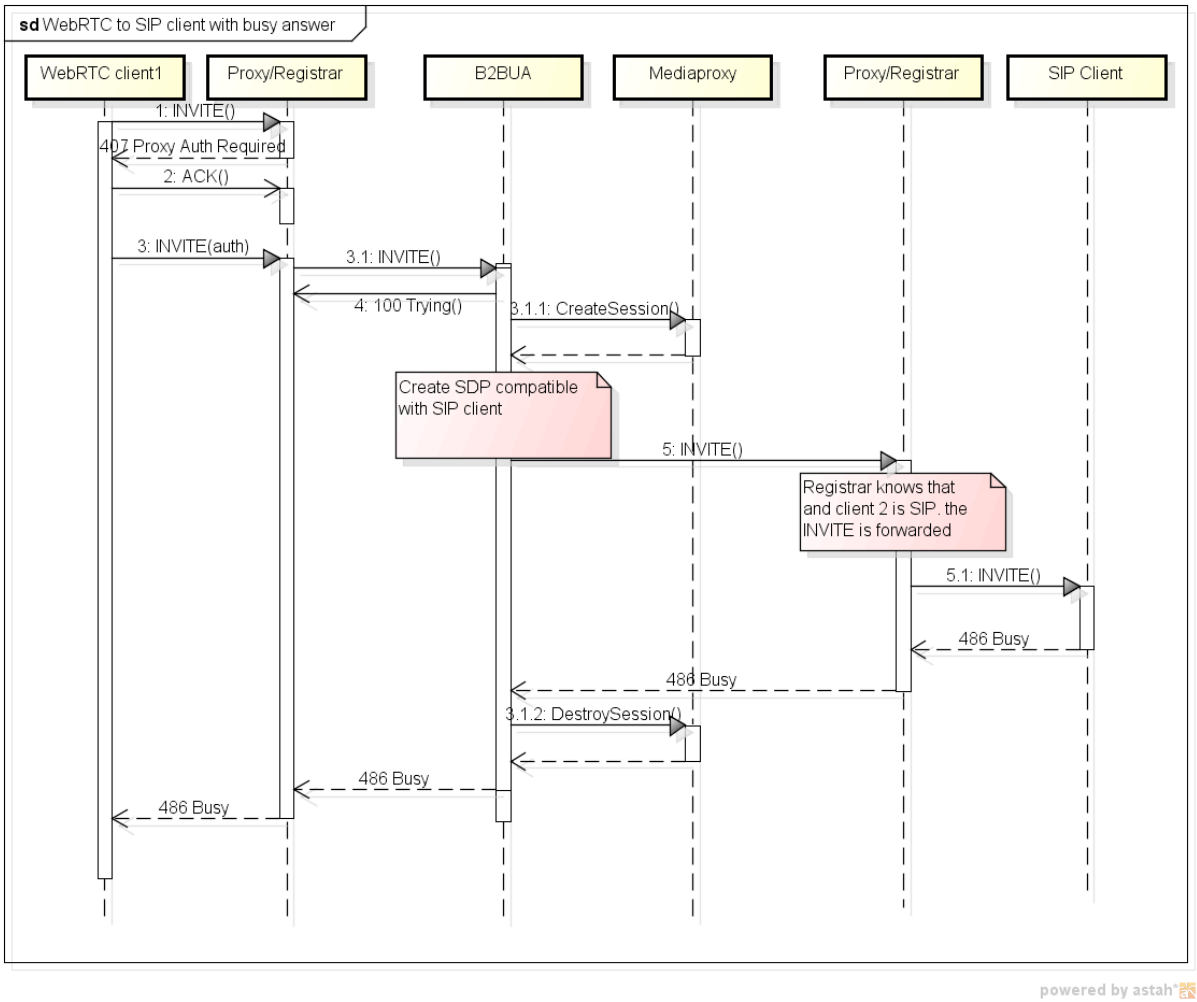


4.4.1 WebRTC to WebRTC call

4.4.2 WebRTC to SIP call



4.4.3 SIP client busy



SIP call to WebRTC

SIP to WebRTC calls

SIP client busy

Late offer / late answer

5 References

T.140 ITU-T T.140
RFC 4103
RFC 7118
RFC 3261
RFC 4467
RFC 4566
RFC 5888