27-04-2016

SA8T2 Internal Deliverable
Technology Scout: WebRTC2SIP Gateway

SA8T2 Internal Deliverable

Contractual Date:	30-04-2016
Actual Date:	27-04-2016
Grant Agreement No.:	691567
Activity:	12/SA8
Task Item:	Task 2 – WebRTC
Nature of Deliverable:	R (Report)
Dissemination Level:	PU (Public)
Lead Partner:	NORDUnet (UNINETT)
Authors:	Stefan Otto (UNINETT)

The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 691567 (GN4-1).

Abstract
This document reports on results and findings from a technical investigation into a WebRTC-to-SIP Gateway. The technology scout was conducted by the Service Activity 8 (SA8, Real Time Communication and Media), Task 2 (WebRTC) team as part of the GN4-1 project. This report should, as such, be read in context of the related work produced by GN4-1 SA8-T2.

Table of Contents

Table of Figures

1 Introduction

1.1 About this document

This document reports on results and findings from a technology scout into a WebRTC-to-SIP Gateway conducted by the Service Activity 8 (SA8), Task 2 team of the GN4-1 project. This report should, as such, be read in context of the related work produced by GN4-1 SA8-T2.

The WebRTC task ran from 1 May 2015 to 30 April 2016.

1.1.1 Target audience

This document targets technical management and specialists, in particular those working in the fields of real time communications, eLearning and eResearch.

1.1.2 Responsible task members

Stefan Otto (UNINETT) had the lead on this tech scout. Jan Meijer (UNINETT) and Simon Skrødal (UNINETT) were the document editors.

1.2 Background

The Session Initiation Protocol (SIP) is a system of rules that governs the signalling and controlling of multimedia communication sessions. This technology scout investigates how the WebRTC ecosystem may integrate and support legacy SIP equipment.

There are a number of scenarios where it would make sense to enable audio/video communication between legacy SIP clients and web browsers. While participation in a classical SIP video meeting with a WebRTC-enabled browser is already possible, the other way around is not. You can not participate in a WebRTC browser meeting using a SIP client. This will never work via a universal gateway because there is no standardized signalling protocol in WebRTC, meaning that you would need to implement your own gateway for each WebRTC-service.

Other, more feasible and value-adding use cases include:

- easy to integrate call buttons where web page visitors can ring to your organization's internal telephony network by pressing a button in their browser
- participate with a browser in a classic audio only conference
- build Point-of-Presence systems which can interact both with classical telephony and legacy video systems
- wherever you want to connect a browser to telephony or legacy SIP video systems

WebRTC and SIP media sessions have many similarities (see Figure 1: WebRTC Signalling and media and Figure 2: SIP Signalling and media). Since WebRTC standardization began, there have been ongoing discussions and efforts towards maintaining compatibility with classic SIP audio/video equipment. For this reason, there are no specific signalling protocols defined for WebRTC, meaning you can use SIP if you want. Further, the Session Description Protocol (SDP) is used by SIP as well as WebRTC to describe media session and communication parameters, such as key-fingerprints and IP-address candidates. In WebRTC, this SDP "blob" has to be sent with the signalling protocol of your choice. Hence, there are several similarities worth investigating further in order to bridge WebRTC and SIP.

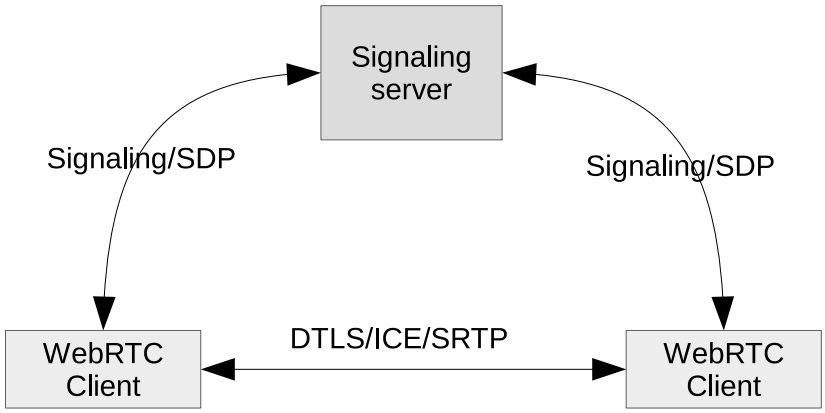

Figure 1: WebRTC Signalling and media

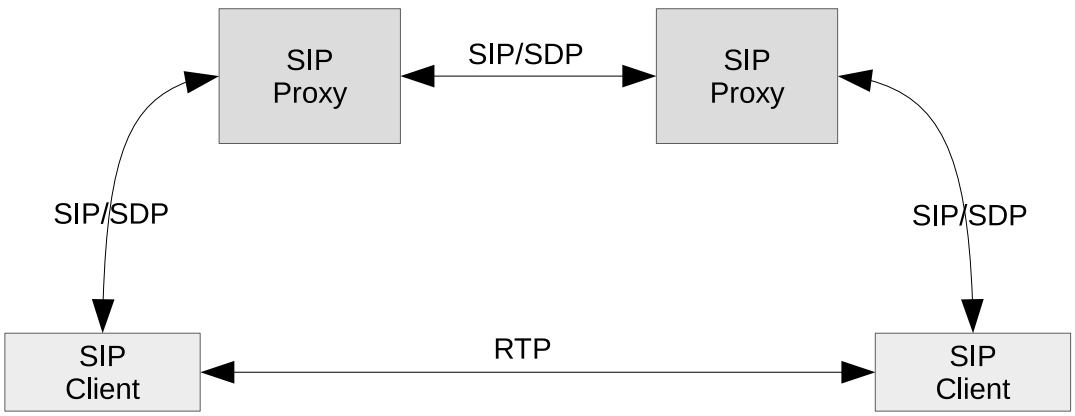

Figure 2: SIP Signalling and media

1.3 Rationale

There is a large installed base of SIP infrastructure in the R&E community. Effectively every Unified Communication installation adds to this. That makes it very interesting to investigate how WebRTC can be used to add value to this installed base, and how the installed base can be used to facilitate WebRTC adoption.

1.4 Tech scout objective and methodology

The objective for this tech scout was to investigate how the WebRTC ecosystem may integrate and support legacy SIP equipment and what type of value-adding use cases could be addressed using the solution(s) identified.

The concept and architecture of a WebRTC2SIP gateway was explored upon which a PoC was built using open source components.

2 Challenges

A closer look identifies a number of issues that must be addressed.

2.1 Transfer Protocol

SIP-clients use direct TCP or UDP to transport SIP packets. The web browser, on the other hand, can only communicate using HTTP/WebSocket. Since the web developer cannot make the web browser send or receive SIP-messages to/from a specified IP-address/port, the transfer protocol must be converted. This may be achieved by using the WebSocket protocol as transport for SIP (see https://tools.ietf.org/html/rfc7118) or to translate to SIP on the WebRTC-signalling server and send forwards.

2.2 Codecs

The next challenge involves the video/audio codec negotiation. Current WebRTC-enabled browsers (e.g. Firefox/Chrome) support the following codecs:

Video — VP8 (MTI), VP9 and H264 (MTI - supported on Firefox, experimental support in Chrome-M50)

Audio — opus (MTI), G.711 (MTI), iSAC and G.722

SIP clients usually understand several codecs, but if there is no single match to the ones from the browser they will not be able to talk to each other. A gateway could decode and encode the streams to the supported codec, but this would add significant CPU workload, additional latency and compatibility issues.

2.3 Encryption

For WebRTC connections it is mandatory to use DTLS to establish a secure SRTP connection. Only a few legacy systems support this.

2.4 ICE

In order to establish connections through firewalls and NAT, routers, WebRTC clients and SIP-clients use ICE (Interactive Connectivity Establishment, see https://tools.ietf.org/html/rfc5245). There exist several improving extensions, e.g. TCP Candidates with ICE (https://tools.ietf.org/html/rfc6544) and the Trickle ICE draft (https://tools.ietf.org/html/draft-ietf-mmusic-trickle-ice-02), which legacy SIP equipment usually does not know about, but WebRTC clients do.

Other issues include RTP/RTCP muxing into one UDP port (http://tools.ietf.org/html/rfc5761) and bundling several media streams in one address:port combination (BUNDLE draft — https://tools.ietf.org/html/draft-ietf-mmusic-sdp-bundle-negotiation-27), used by WebRTC-clients and preventing direct communication with SIP legacy clients.

3 Possible candidates

A possible gateway architecture is suggested by Figure 3: WebRTC2SIP gateway.

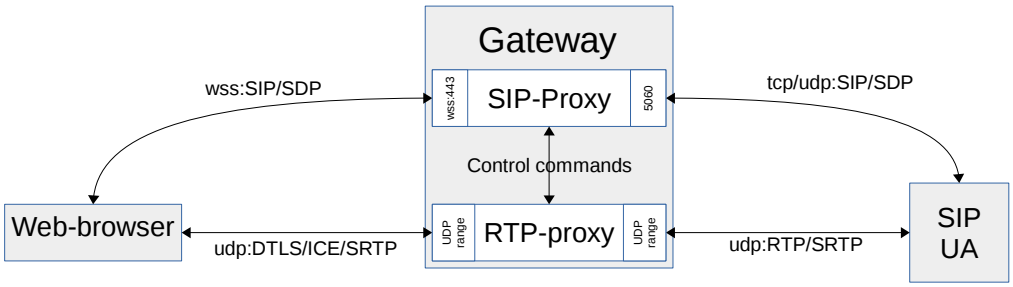

Figure 3: WebRTC2SIP gateway

The gateway consists of a SIP-proxy and an RTP-proxy. The SIP-proxy accepts SIP packets via the web client's WebSocket. SDP is translated (delete, generate, replace ICE candidates, BUNDLE lines, DTLS fingerprints, ...) and the RTP-proxy then bridges media traffic (IPv4/IPv6, SRTP/RTP, RTP/RTCP multiplexing/demultiplexing, BUNDLE/unBUNDLE media streams, ...).

Since the technology scout is restricted to consider open source solutions only, several *commercial* solutions were disregarded. However, for the sake of completeness, we want to mention Acano (https://www.acano.com) and Pexip (https://www.pexip.com), which include WebRTC2SIP gateway functionality in their products.

3.1 FreeSWITCH

While FreeSWITCH (https://freeswitch.org)looks like a promising candidate worth testing, we were unable to do so due to time constraints.

It features a scalable telephony platform designed to route and interconnect popular communication protocols using audio, video, text or any other form of media, support communication technologies such as Skype, SIP, H.323 and WebRTC. FreeSWITCH can perform full video transcoding and MCU functionality using its conferencing module.

3.2 Doubango webrtc2sip

Doubango Telecom is a young telecom company located in France, with a big focus on open source solutions in IMS/SIP and WebRTC area. SIPml5, a browser JavaScript SIP library, is provided by Doubango Telecom.

Their Doubango webrtc2sip media gateway (https://www.doubango.org/webrtc2sip/) consists of 4 modules (SIP Proxy, RTCWeb Breaker, Media Coder, Click-to-Call) following the same philosophy as described in this document. Additionally, it features audio/video transcoding abilities. In our tests one year ago we experienced the solution as slightly unstable. There are no binary packages, so you have to build everything on your own. After updating openSSL due to security issues, we were no longer able to build it and thus ended our testing of webrtc2sip at that point.

3.3 Janus

The Italian company Meetecho, located in Naples, provides an open source general purpose WebRTC Gateway called Janus (https://janus.conf.meetecho.com). Janus consists of a small footprint core, which can be extended/customized with a number of plugins. There are already a few plugins available, e.g. audiobridge, echotest, recordplay, SIP, streaming, videocall, videoroom and voicemail. The SIP plugin enables peers to register through Janus to a SIP server and call in and out. The functionality is marked as unstable and need further testing - for now tested successfully audio only.

The SIP plugin is based on Sofia SIP (http://sofia-sip.sourceforge.net). No documentation was found in regard to SDP translating between legacy SIP equipment and WebRTC.

3.4 Kamailio + rtpengine

The author of this document is part of the Norwegian real-time group at UNINETT, and we built a SIP infrastructure based on the solid SIP router Kamailio (https://www.kamailio.org). It therefore made sense to focus our attention on a SIP-proxy solution based on Kamailio.

4 Solution Overview

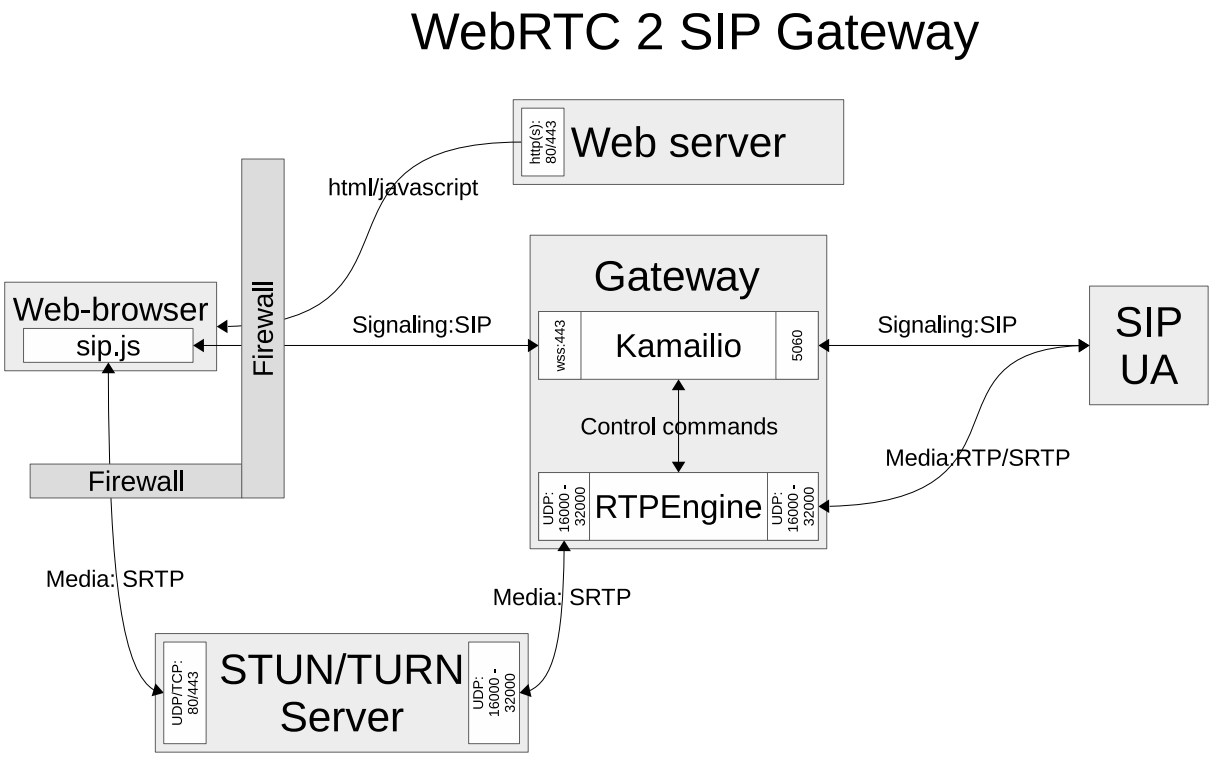

Figure 4: WebRTC2SIP-Gateway overview

The implemented solution utilizes Kamailio as the SIP-proxy and Sipwise rtpengine (https://github.com/sipwise/rtpengine) as the RTP-proxy. Kamailio controls rtpengine with the integrated rtpengine module to facilitate the following: Kamailio passes a received SDP body to rtpengine → rtpengine rewrites the body and returns it → Kamailio forwards the rewritten body.

This flow guarantees that each SIP-client (WebRTC SIP.js clients as well) gets only ICE candidates from rtpengine. In result RTP traffic will flow via rtpengine. Additionally, rtpengine takes care of BUNDLEd media and muxed RTP/RTCP channels and rewrites these lines in the SDP body as well. Hence, rtpengine effectively talks to each client side and translates in the middle (this does not include media transcoding).

The web server is the entry point for the web browser and its naming should be user friendly (e.g. call-the-ancient-world.net). Running the HTML/Javascript-code from the web server turns the browser into a WebRTC SIP client that registers on Kamailio. The client is then callable via a SIP-URI world-wide or can place its own calls. In this example configuration, the SIP-URI is websip@gateway-domain.no

Should the WebRTC client be unable to connect directly to rtpengine it will receive support from the TURN-server, which listens on TCP/TLS port 443 and forwards the media traffic to rtpengine via UDP. This is needed to pass restrictive firewalls on the client side. rtpengine does not support RTP media over TCP for now — if this will be the case in the (near) future, there will no longer be a need for a TURN server.

All components could be located on the same server, but in order to get listening on port 443 working for all servers for firewall traversal, web-server, Kamailio and TURN server each will need an independent IP-address.

5 Limitations

rtpengine is just a proxy so there is no media transcoding. SIP clients that do not support VP8/H264 video codecs cannot be called or make calls. For now, only Firefox supports both video codecs, Chrome's support for H264 is not stable yet and Microsoft Edge supports only H264. As the IETF rtcweb-standardization group has decided to make both video-codecs mandatory to implement, I sooner or later expect support for both codecs in all WebRTC-enabled browsers. On the other side, legacy SIP clients usually support H264. Some modern systems and software clients integrate VP8 as well. Acano, for example, has both on their system.

6 Performance and Scalability

Since there is no media transcoding, the limiting factor would be the network bandwidth for media traffic over rtpengine. For large scale installations, rtpengine could be distributed over several instances on different locations. For a 1 Gigabit Ethernet interface there should be roughly 100 concurrent calls possible.

7 Security considerations

User credentials for the SIP-proxy and for the TURN server are publicly available from the web server. Some kind of authentication should therefore be established in front of the web server. The TURN server should be protected with time-limited secret-based authentication (see A REST API For Access To TURN Services, https://tools.ietf.org/html/draft-uberti-behave-turn-rest-00) or an oAuth-based client authorization, which is experimentally supported by the coTURN TURN server. To protect Kamailio, we could generate user accounts dynamically on the web server and push these to Kamailio's user-database.

Since media traffic is decrypted in the Gateway, it is possible to intercept the media traffic between the gateway and legacy SIP equipment. Metadata interception is possible if you intercept network traffic to and from the gateway. You can get IP-address based call information (when, how long, SIP header information).

8 Conclusions

We have demonstrated that it is possible to build a scalable WebRTC gateway from open source components. It was possible to get stable connections to:

- Alcatel-Lucent PSTN-Gateway (audio)
- Asterisk based conference system (audio)
- Several softphones such as Ekiga or Linphone (audio + video)
- Acano video meeting (audio + video)

The solution is far away from being interoperable with the whole SIP legacy world, but it is a starting point. There is still a video codec problem, which hopefully will be solved soon. There are also issues with several SIP systems that must be analyzed further. However, it is already possible to deliver a stable "call button" solution, or an alternative video client, for calling modern videoconferencing systems like Acano.

Appendix A Setup WebRTC2SIP-gateway

This section describes how to install and set up Kamailio + rtpengine + TURN server + WebRTC client to enable calling between WebRTC clients and legacy SIP clients. This setup will bridge SRTP-RTP and ICE-nonICE to make WebRTC clients (SIP.js) able to call legacy SIP clients.

The setup is pertinent to Debian 8 Jessie for all servers.

For clients to avoid firewalls and to have the best setup, divide the servers like this:

1. **Server** — Kamailio + rtpengine
2. **Server** — TURN (provide TCP/UDP connections on port 443 for your WebRTC clients to pass restrictive firewalls)
3. **Server** — Web server to serve the WebRTC client with SIP.js

The configuration is set up to try connecting with SIP without modification. If the proxy receives a "488 Not Supported Here" from the other side, it will remove SRTP and ICE from the SDP and try again to INVITE.

A.1 Get certificates

For the needed certificates, I recommend Let's Encrypt (https://letsencrypt.org). They will work for Kamailio TLS, Nginx TLS and coTURN TLS. Run the following on each server you need certificates for (you must stop services running on port 443 during certificate request/renewal):

```
# git clone https://github.com/letsencrypt/letsencrypt
# cd letsencrypt
# ./letsencrypt certonly --standalone -d YOUR-DOMAIN
```

You will then find the certificates under:

```
/etc/letsencrypt/live/YOUR-DOMAIN/privkey.pem
/etc/letsencrypt/live/YOUR-DOMAIN/fullchain.pem
```

Copy them to the appropriate location.

A.2 Get configuration files

Files needed to setup all components on Debian 8 Jessie:

```
git clone https://github.com/havfo/WebRTC-to-SIP.git
cd WebRTC-to-SIP
find . -type f -print0 | xargs -0 sed -i 's/XXXXX-XXXXX/PUT-IP-OF-YOUR-SIP-SERVER-HERE/g'
find . -type f -print0 | xargs -0 sed -i 's/XXXX-XXXX/PUT-DOMAIN-OF-YOUR-SIP-SERVER-HERE/g'
find . -type f -print0 | xargs -0 sed -i 's/XXX-XXX/PUT-DOMAIN-OF-YOUR-TURN-SERVER-HERE/g'
```

A.3 Install rtpengine

The following achieves the SRTP-RTP bridging needed to make WebRTC clients talk to legacy SIP server/clients:

```
apt-get install dpkg-dev debhelper iptables-dev
      libcurl4-openssl-dev libglib2.0-dev libhiredis-dev
      libpcre3-dev libssl-dev markdown zlib1g-dev
      libxmlrpc-core-c3-dev dkms linux-headers-`uname -r`
git clone https://github.com/sipwise/rtpengine.git
cd rtpengine
./debian/flavors/no_ngcp
dpkg-buildpackage
cd ..
dpkg   -i   'ngcp-rtpengine-daemon_*'   'ngcp-rtpengine-iptables_*'   'ngcp-rtpengine-kernel-dkms_*'
cd WebRTC-to-SIP
cp etc/default/ngcp-rtpengine-daemon /etc/default/
/etc/init.d/ngcp-rtpengine-daemon restart
```

A.4 Install IPTables firewall (optional)

Optional, but recommended to secure your servers. It will work on all three servers, but you should customize _iptables.sh_ for each of them (e.g. the web server does not need the TURN, RTPENGINE and RTP rules, while the TURN server does not need the RTPENGINE rule). If installed it will persist after reboot. You can run the _iptables.sh_ at any time after it is set up.

```
cd WebRTC-to-SIP
chmod +x iptables.sh
cp etc/network/if-up.d/iptables /etc/network/if-up.d/
chmod +x /etc/network/if-up.d/iptables
touch /etc/iptables/firewall.conf
touch /etc/iptables/firewall6.conf
./iptables.sh
```

A.5 Install Kamailio

```
apt-get install kamailio kamailio-websocket-modules kamailio-mysql-modules
kamailio-tls-modules kamailio-presence-modules mysql-server
cd WebRTC-to-SIP
```

```
cp etc/kamailio/* /etc/kamailio/
kamdbctl create
```

Select yes (Y) for all options.

```
kamctl add websip websip
/etc/init.d/kamailio restart
```

A.6 Install WebRTC client

```
apt-get install nginx
cd WebRTC-to-SIP
cp etc/nginx/sites-available/default /etc/nginx/sites-available/
cp -r client/* /var/www/html/
```

A.7 Install TURN server

```
apt-get install rfc5766-turn-server
cp etc/default/rfc5766-turn-server /etc/default/
cp etc/turn* /etc/
/etc/init.d/rfc5766-turn-server restart
```

A.8 Testing

You should now be able to go to https://webrtcnginxserver/ and call to legacy SIP clients.