Real-Time Communications Quick Start Guide

Daniel Pocock [http://danielpocock.com]

Real-Time Communications Quick Start Guide

Daniel Pocock [http://danielpocock.com]

Table of Contents

List of Figures

List of Tables

List of Examples

Preface

Mission

This book aims to provide practical strategies for deploying RTC with the technology available today.

Vision

A world where open standards and free software are the foundation of personal and business communications, enabling genuine innovation and the emergence of more disruptive technologies.

Who is this document for?

IT managers, system administrators, developers, web designers, product managers and IT users who want best-practice Real-Time Communications (RTC) technology for business or private use.

What is Free RTC?

Running your own, independent, federated and peer-to-peer RTC solutions, including instant messaging (IM), voice-over-IP (VoIP), video/webcam, social networking and WebRTC, using open standards and, in many cases, free, open source software.

Why?

There are many reasons organizations and individuals need to run their own RTC infrastructure:

Resilience: operating RTC servers to the same high standard as the rest of your non-stop infrastructure rather than relying on some vendor who provides a free download for anybody and everybody.

Security: avoid installing proprietary, third party communications apps and plugins.

Privacy: avoid letting sensitive information be harvested by cloud providers.

Brand building: keep users on your own web site, assert your domain name in all communication sessions.

Control: recognize callers who are already logged in to your web site and route their call efficiently based on language, account size or other factors.

Innovation: in traditional phone companies, technical innovation has slowed. Open standards and free software allow individuals and businesses of any size to engage in genuine innovation, creating new and original services that run across the network.

How?

This documentation aims to help you choose strong, best of breed, stable and supported components based on genuinely free software [https://www.gnu.org/philosophy/free-sw.html] and open standards. There are step-by-step instructions for DNS, firewall and server configuration and testing to achieve maximum chance of success for every call or chat connection your users need to make.

Thanks to the convenient packages in Linux (Debian, Ubuntu, Fedora and Red Hat/RHEL/CentOS), most IT professionals will be able to set this up in less than one day, the most experienced reader will find that it can be set up in less than an hour.

Chapter 1. Introduction

This quick start guide walks through the essential steps to build a working real-time communications platform with full support for federation with other autonomous domains over the public Internet.

We show the essential steps first: setting up a TURN server, SIP proxy and an XMPP server. Setting up an Asterisk or FreeSWITCH PBX is not essential, these are supplementary services that should be added in a later stage of the project.

Federation

Federation enables direct and efficient communication between any two autonomous organizations connected to the public Internet. Email is already distributed thanks to SMTP federation. The same paradigm has taken hold in the world of voice and video communications. Any two users or organisations can connect to each other dynamically without requiring cumbersome, outdated and expensive solutions from traditional phone companies.

A number of technologies help make federation work optimally. ENUM helps map traditional phone numbers to Internet domains, it is described in Chapter 15, *Adding ENUM to DNS*. DNS NAPTR and SRV records make it possible to locate servers willing to accept calls any given domain, they are described in Chapter 5, *DNS setup*. The SIP and XMPP protocols allow users to do much more than they can do with traditional phones, including cost-effective chat messaging, desktop sharing and videoconferencing.

Independent and decentralized alternatives to federation

Federation takes a lot of power from the telephone industry and places more power and control in the hands of organizations and individuals who run their own servers. This is generally a good thing for innovation, cost-efficiency, privacy and many other reasons.

Critics of federation observe that while this model is not as tightly centralized as a traditional telephone network, it is built around a client-server model, leaving power in the hands of those who are able to operate the servers. Like many Internet-based technologies, it also relies heavily on other centralized services: the DNS protocol and the certificate authorities.

Private networks

Some operators have created private networks, where users can only call other users with the same softphone. All the users communicate through a central server. The operator chooses to locate the server in a location they believe to be secure and where they believe the risk of surveillance is low, such as Switzerland.

Signal [https://en.wikipedia.org/wiki/Signal_(software)], the successor to RedPhone and TextSecure from Open Whisper Systems, operates through privately run servers. Administrators of the servers are able to observe who is calling who but without knowing what they are saying.

Decentralized networks

Various solutions have begun to emerge in the hope of further eliminating these dependencies and offering a genuinely decentralized service. In a truly decentralized service, the developer/founder of the service is not even able to see which users are calling each other, making it more secure than privately run services such as Signal.

A fundamental issue for these alternatives is the addressing scheme. Some rely on phone numbers and some of these solutions require their users to use an identifier other than a phone number, exchanging the identifiers in person or though some other communications channel. If phone numbers are used, it makes the service more convenient but this implies placing some trust in the phone numbering scheme operated by the telephone industry.

Another fundamental issue for these services is bootstrapping: how the client finds other peer-to-peer participants the first time the user runs the software. The current solution to this is usually a central server keeping a list of peers to seed the clients.

Examples of some decentralized and peer-to-peer networks include the Ring softphone from Savoir Faire Linux [https://ring.cx], using the OpenDHT [https://github.com/savoirfairelinux/opendht] network, the Tox app [http://tox.chat/] and the Matrix.org [https://matrix.org/] service.

Conclusion

While some of these alternatives are promising, none of them offers a silver bullet. Private and decentralized services are only useful for specific purposes where two people know each other personally, such as calling a spouse, a physician calling a patient or a lawyer communicating with a client.

For large organizations that deal with other large organizations and with the public in a less personal manner, the decentralized model is not universally applicable and a federated model is more likely to gain traction and meet operational needs. That said, it is not uncommon for senior executives in some large corporations to seek out specialized communications solutions so they can have private conversations with each other and their closest advisors.

Decentralized services are not mutually exclusive with federated RTC. It is quite feasible for an organization to operate standard SIP or XMPP internally but setup a gateway at the edge of their network to accept calls from customers using alternative services. The external user needs to have some way to be certain that their call is connected to an account controlled by the organization they want to contact and not an imposter or a man-in-the-middle who is relaying the call while monitoring it.

Choosing between SIP and XMPP

The IETF has documented two standards for real-time communications that are widely implemented: SIP and XMPP. XMPP is sometimes referred to as Jabber.

This has left many system administrators pondering the question: which one do I need?

Both SIP and XMPP can do all the same things. SIP can make phone calls, video calls and instant messaging (IM) sessions. XMPP can also make phone calls, video calls and IM sessions. There has been a tendency to use SIP more for voice and use XMPP more for IM.

Both have evolved into different markets. SIP is particularly well supported by telecommunications vendors (for example, companies offering trunking to or from the PSTN) and manufacturers of related hardware, such as ISDN gateways. Many larger corporate phone systems also have some kind of SIP interface. XMPP has become very prominent as a system for federated IM and many companies have deployed it for this purpose without necessarily using it for voice and video. Just as a significant number of voice related products and services use SIP, there is a significant quantity of high quality IM client software and third-party frameworks for interaction with XMPP and this has helped it remain dominant in the IM domain.

In many cases, a SIP user ID and an XMPP user ID look identical, except for the URI prefix and both can be used to reach the same physical person. For example, `sip:alice@example.org` and `xmpp:alice@example.org` both provide a means to contact the same fictitious Internet user prominent in so many of the IETF's publications.

From the user's perspective, when they see an address without a scheme prefix such as `sip:` or `xmpp:`, they have no way to know if it is useful for email, SIP or XMPP and may have to manually try it in several applications. Some users may not even realize that these different protocols exist for RTC and if the address doesn't appear to work in the first application where they try to use it, they make come to the conclusion that the address is invalid or the other person is unreachable.

The overwhelming recommendation of the author of this guide and the software described here is that to maximise your chance of communicating with as many users as possible, *you should operate both SIP and XMPP servers in parallel*.

Fortunately, some of the infrastructure for these servers (such as TURN servers and the X.509 certificates) can be shared by both protocols.

Choice of operating system

RTC is possible using a range of operating systems, including those popular on desktop computers, servers and smartphones. This guide does not recommend a specific operating system. However, we believe that for people who want to mix and match individual packages, the most recent stable releases of popular GNU/Linux distributions, including Debian, Ubuntu and Fedora, provide a convenient way to get all the necessary software in ready-to-run packages. For people who want a turn-key solution, it may be better to choose one of the self-installing or ready-to-run RTC or groupware solutions.

Using a ready-to-run or turn-key solution

There are various turn-key solutions for building servers for RTC or generic office/groupware purposes. These typically run off a live ISO image, provide a script to pre-install and configure packages or they are an image for a platform like Docker.

Examples of these platforms include WikiSuite [http://WikiSuite.org] (based on RHEL) and Turnkey Linux [https://www.turnkeylinux.org/] (based on Debian).

Using a generic GNU/Linux distribution

Users of Red Hat Enterprise Linux, CentOS, openSUSE, SLES and other platforms may need to build the packages manually using `rpmbuild` as described in Appendix B, *Building reSIProcate RPMs on RHEL and CentOS*.

Several of the components described in this guide have been tested on a much wider range of platforms. The *reSIProcate* products, including the *repro* SIP proxy and *reTurn* TURN server, are extremely versatile and known to run successfully on Microsoft Windows, Apple OS, iOS, BSD variants, Android and several Linux based routers including OpenWRT, CeroWRT and DD-WRT.

Use latest software versions

It is recommended that the latest software versions are used, especially for components such as the TURN server, SIP proxy and XMPP server as these components need to achieve connectivity with a wide range of peers on the public Internet.

This does not imply using an unstable or beta version of your preferred Linux distribution, such as Debian *sid* or Fedora *rawhide*. Rather, it is recommended that the current stable release of the operating system is used and the RTC components can then be installed from a source such as Debian's *stable-backports* or Red Hat's *EPEL*.

Sometimes *stable-backports* or *EPEL* won't have the latest version of a particular package or you want to test some bleeding edge version of the package to see if it fixes a particular bug. Many of the packages can be built manually from the source code. All the leading RTC server projects make this very easy as they support tools like `debuild` for Debian/Ubuntu (see Appendix A, *Building reSIProcate packages on Debian/Ubuntu*) or `rpmbuild` for RedHat/CentOS/Fedora (see Appendix B, *Building reSIProcate RPMs on RHEL and CentOS*).

Using IPv6

Now that IPv4 address space is fully allocated, it is highly desirable for organizations to include IPv6 when implementing any new network services.

Therefore, both IPv4 and IPv6 are used in all examples throughout this guide.

All of the recommended software products work on both IP versions.

Everything in this guide will still work even if you only use IPv4 or IPv6 alone.

Example network used in the documentation

For the purposes of this guide, the following conventions are used:

The DNS domain name is *example.org*.

All the applications run on a server called *server1.example.org*. In practice, you could run each service on a different server and you may duplicate services across multiple servers for *N+1* redundancy.

The ICE/STUN process requires two public IPv4 addresses, either on the same interface or on different interfaces. In the examples, the server has two IPv4 addresses on the same interface/subnet, they are *198.51.100.19* and *198.51.100.20*. These IP addresses come from the RFC 5737 documentation subnets [https://tools.ietf.org/html/rfc5737].

For IPv6, RFC 3849 reserves the address prefix *2001:DB8::/32* for documentation. In the examples, *server1.example.org* uses the address *2001:DB8:1000:2000::19/64*.

The users have existing email addresses such as *first.last@example.org* and will use the same addresses for both SIP and XMPP.

The internal phone numbers are four digit extensions, such as *8001*, *8002* and *8003*.

Chapter 2. Architecture overview

This chapter gives a high-level overview of the RTC architecture. Each component is explained in more detail in its own chapter.

The big picture

Figure 2.1. Overview

Figure 2.1, “Overview” demonstrates each of the components and how they are interconnected. The diagram includes an example of an external softphone user calling an internal softphone user, the call is setup with SIP and the RTP media streams (dotted lines) pass through the TURN server.

TLS is essential

SIP, XMPP and WebSockets can be easily configured to run without TLS encryption. Unfortunately, doing so would lead to many of the same problems as email, including spam and impersonation.

Impersonation is even more troublesome in RTC than in an email exchange. If a user replies to an email with a forged `From` header, the reply will go to the person who was impersonated. The imposter is unable to receive replies to the emails they send. If a user answers a phone call from a forged SIP address, however, they are immediately engaged in two-way communication with the imposter.

Therefore, when RTC protocols are used on the public Internet, TLS should always be used. Additional reasons for using TLS are discussed in the section called “Use the TLS transport for SIP signalling”.

SMTP is a much older protocol than SIP and XMPP and while it does now boast support for STARTTLS, it doesn't clearly specify a mechanism for validation of message headers against the certificates [http://tools.ietf.org/html/rfc6125#appendix-B.4].

All SIP connectivity through a SIP proxy

The SIP proxy acts as a router between the external peers, internal peers and the soft PBX. The soft PBX is typically a server running Asterisk or FreeSWITCH. It is important to note that the soft PBX does not connect directly to the public Internet and none of the internal users connect directly to the soft PBX.

SIP proxy servers are generally more stable and more secure than soft PBXes. SIP proxy servers typically have more connectivity options, including best-of-breed support for IPv6, TLS and WebRTC. In particular, the Asterisk PBX advertises support for TLS but it doesn't support mutual TLS certificate verification, something that works seamlessly in the SIP proxy *repro*. This means that Asterisk accepts TLS connections from users and other servers but is unable to verify local devices with built-in certificates such as Polycom phones. If Asterisk is configured to accept TLS connections from the public Internet, Asterisk accepts any call from the peer without validating the domain in the `From` header.

Soft PBXes tend to have many more features and vastly more configuration options, this also means upgrades to the SIP proxy are relatively easy compared to upgrades of the soft PBX. Finally, some people like to be able to make configuration changes to their PBX during business hours. If users are maintaining connections and SIP registrations through the SIP proxy, they are much less likely to notice if the soft PBX is restarted or crashes.

One consequence of this design strategy is that it is usually best to install, test and configure the SIP proxy before starting a soft PBX installation. In this guide, SIP proxy installation is covered in Chapter 11, *SIP proxy server installation* and soft PBXes are discussed in Chapter 17, *PBX Setup*.

SIP federation between two autonomous sites

Figure 2.2. SIP federation between two sites

Figure 2.2, “SIP federation between two sites” emphasizes those components that are involved in routing a federated SIP call from one site, *example.org*, to another site, *example.edu*. For simplicity, this diagram does not show the firewalls, soft PBX or other components. Assuming the softphone users are both using NAT addresses, the TURN servers may be relaying all the media streams on their behalf.

Routing calls within a site

Figure 2.3. Four stages of call routing

If you have just started looking at the configuration of a SIP proxy or soft PBX, you have probably observed that the scripting languages provide almost infinite flexibility to route the calls in different ways. If you have been working with such configurations for a while, you may have seen some that have become tremendously convoluted.

Figure 2.3, “Four stages of call routing” demonstrates a high-level approach to routing calls within your site, whether you are using a single SIP proxy or soft PBX or a combination of different components.

There are four general stages.

All calls, whether they come from SIP trunking providers, ISDN or local users should start in an *ingress* stage. The goal of the *ingress* stage is simply normalizing the numbers into a standard form that will be useful in all further stages of call processing. For example, if calls are coming in over an ISDN connection the provider may only be sending the last six digits of each destination DDI that has been dialed. The *ingress* stage handling calls from this ISDN circuit adds the country code and other leading digits to normalize the numbers in the E.164 format (see Chapter 18, *PSTN connectivity* for more specific details about this type of *ingress* handling).

When calls are sent to their final destination, whether it is a SIP trunking provider, ISDN circuit or a local user, the final stage it should go through is an *egress* stage. The format of the number/address usually needs to be modified again in the *egress* stage. For example, some providers expect E.164 numbers to have a 00 prefix, as this is the dialing prefix many countries use to call international numbers.

Many deployments involve some services where calls are handled by an application. This is the *application* stage. These applications are typically voicemail services, call queues and DTMF-driven menus.

Finally, all these stages are joined together by a *routing* stage. The routing stage accepts calls from the *ingress* stage, considers both the source of the call and the desired destination (both of which have been normalized already) and decides where to send them in the *egress* and *application* stages.

All these stages can be implemented in a single system such as an Asterisk PBX. For ease of administration, it is advisable to break the `extensions.conf` file up into different files for each stage as demonstrated in Example 2.1, “Splitting Asterisk `extensions.conf`”.

The more powerful approach is to transpose this paradigm over several individual processes and devices. For example, the *ingress* stage for calls from local users may be implemented in the SIP proxy and the *ingress* stage for calls from an ISDN circuit may be implemented using the configuration file in a media gateway.

Example 2.1. Splitting Asterisk `extensions.conf`

```
#include "/etc/asterisk/extensions/ingress/local_users.conf"
#include "/etc/asterisk/extensions/ingress/trunks.conf"
#include "/etc/asterisk/extensions/routing.conf"
#include "/etc/asterisk/extensions/applications.conf"
#include "/etc/asterisk/extensions/egress/local_users.conf"
#include "/etc/asterisk/extensions/egress/trunks.conf"
```

WebRTC peer-to-peer calling

Figure 2.4. WebRTC basic peer-to-peer

Figure 2.4, “WebRTC basic peer-to-peer” demonstrates how two browsers can communicate with each other using WebRTC. The web browsers start by downloading the HTML, CSS and JavaScript from a normal web server such as Apache `httpd`. The JavaScript uses the WebSocket protocol to initiate a connection to the SIP proxy. When a call is made, the request is sent over the WebSocket connection and the media streams pass through the TURN server. In this case, the browsers are not relaying the media streams through the TURN server, possibly because they have discovered they are both on the same IP network.

WebRTC calling to call centers

Figure 2.5. WebRTC from customer web browser to call center

Figure 2.5, “WebRTC from customer web browser to call center” demonstrates a more elaborate WebRTC architecture, a customer using a web browser to call a corporate call center. When a call is made, the request is sent over the WebSocket connection and the media streams pass through the TURN server. The SIP proxy routes all calls to the corporate PBX which routes the calls to an agent. The media streams must also pass through the PBX for transcoding from the Opus codec to one of the codecs supported by the desk phones, perhaps G.711 or G.722.

Chapter 3. User Experience

Any successful IT project needs to focus on the needs of the user. This is particularly important for communications technology.

First time setup and provisioning

The most successful RTC projects all have a convenient means of user provisioning.

In an office environment, this may mean that the softphone or desk phone is automatically configured for the user by a system administrator. Chapter 14, *Client devices and softphones* discusses the provisioning facilities of some types of phone.

For a generic SIP service, an ISP or a corporate/campus environment with a bring-your-own (BYO) device policy, the ideal setup process should only require the user to enter their SIP/XMPP address and their password and the softphone/device should discover all other parameters using DNS NAPTR and DNS SRV queries. For this to be effective, softphones and devices aiming to be used in this way need to ensure they have suitable codec and encryption settings enabled by default so the user won't have to tweak them as recommended in Chapter 4, *Optimizing Connectivity*.

Dialing

At first glance, dialing a telephone may appear to be a trivial task. Some attention to detail is required to maximize user convenience.

Usernames or phone numbers?

A user may want to call or chat to various people. For some people, they may only have the email/SIP/XMPP address and for other people they may only have a phone number.

It is important to select phones that work intuitively for either type of input, an address of the form `user@example.org` or a phone number. It is also important to ensure that the server processes, such as SIP proxies, are correctly configured to route calls to arbitrary Internet addresses. The repro SIP proxy and most XMPP servers work this way by default.

Some phones can even help convert from a phone number to a SIP or XMPP address, see Chapter 15, *Adding ENUM to DNS*.

If the user has an entry in their address book with both a phone number and an email/SIP/XMPP address, then it is important that the phone gives the user a helpful way to choose which one to dial without asking too many questions about whether the user wants to use SIP or XMPP. This process can be optimized if the phone uses presence (a buddy list) to work out which contact mechanisms are unreachable.

All of this can be made easier by using named accounts (like email addresses) instead of extension numbers within the server configuration, this is explained in more detail in the section called “Personal account names or extension numbers”.

Dial plans

Users find it easier to dial numbers in a local format (without a country code). Many personal address books and company databases store phone numbers in a local format. This can lead to confusion in situations where somebody tries to use a number in a different country, for example, if they take their mobile/cell phone to another country and try to dial a number stored in the address book.

A dial plan should be designed to convert phone numbers to the international format (E.164), even if a user dials the number in a local format. This makes it easier to send some calls to carriers in different countries and it makes life easier for organizations that expand into multiple countries. This means the user can dial in the local or international format but the phone system will work either way.

Dialing Internet addresses

In many cases, an email address may also be a SIP address or XMPP address. Ideally, when a user tries to dial a contact from their phone, the software should identify whether the contact is reachable over SIP or XMPP and use that route for the call if appropriate.

Chapter 4. Optimizing Connectivity

People who have tried many of the free RTC softphones have observed that they don't always work through firewalls or NAT networks. Sometimes these problems give visual feedback, in the form of error messages advising that the call can't proceed. In other cases, the call appears to be connected but audio only works in one direction or stops after some brief period of time.

Metcalfe's law tells us that the value of a telecommunications network is proportional to the square of the number of connected users of the system (n^2), demonstrated in Figure 4.1, “Metcalfe's law”.

Figure 4.1. Metcalfe's law

Therefore, the benefit of making the solution work for all those users who may suffer in certain NAT environments does not just have a gradual or linear impact, the benefit is quadratic.

Today's RTC technology gives us the tools to deal with these problems in the vast majority of cases. This chapter gives an overview of the main concerns.

Codec selection

Codec is a portmanteau of coder-decoder. A codec is an algorithm for encoding an audio or video signal for storage or transmission over a digital communications network. Codecs are responsible for compression of the data stream and may also take some responsibility for error correction, packet loss concealment and silence suppression.

Each device or softphone typically has one or more codec algorithms included. Some software, such as the Asterisk PBX, can support additional codecs with the help of modules or plugins.

The list of codecs supported by a product depends on the age of the product, patents and the type of products it is intended to interact with. Open source solutions generally avoid patented codecs, although unofficial implementations of them can be found online, such as the popular G.729 codec for Asterisk.

Table 4.1. Common codecs

Name	Type	Bitrate (kbps)	Patented	Comments
G.711 (alaw, ulaw)	audio	64	N	Widely supported in phones, WebRTC browsers, ISDN gateways and virtually everywhere else. Quality of traditional phone calls.
G.722	audio	64	N	Supported in most modern software and high quality desk phones. Transmits higher quality wideband audio in the same 64kbps bandwidth used by G.711
Opus	audio	6 - 510	N	Support in more modern software, WebRTC browsers and some very recent desk phones.
G.729	audio	8	Y	A low bitrate codec supported in a lot of older VoIP phones and related hardware. Voice quality less than a standard telephone call.

Name	Type	Bitrate (kbps)	Patented	Comments
G.723.1	audio	5.3, 6.3	Y	An ultra-low bitrate codec support in some older VoIP phones and related hardware. Voices sound very bad.
GSM full rate	audio	13	N	The original codec for GSM mobile telephony. Supported in many older open source products and some VoIP hardware.
iLBC	audio	15	N	A codec developed for Internet use before Opus. Supported in many older open source products but not widely supported in VoIP hardware.
speex	audio	2-44	N	A codec developed for Internet use before Opus. Supported in many older open source products but not widely supported in VoIP hardware.
VP8	video	depends	N	Mandatory part of WebRTC, supported in browsers. Bitrate depends on resolution and frame rate.
H.264	video	depends	Y	Mandatory part of WebRTC, supported in browsers and in many existing video conferencing hardware products.

The person configuring the software or device can typically select which codecs are permitted and also specify which codecs are preferred over others.

When a call is made, the endpoints negotiate to select a codec that is supported at both ends. If both endpoints have no codecs in common, the call is not possible and the user may see a message telling them the call failed. If both endpoints have more than one codec in common, the exact codec selected in this negotiation algorithm depends on the relative priorities specified by the administrators of each endpoint.

Some software has the ability to dynamically change the list of permitted codecs. For example, some mobile apps will only enable high-bandwidth codecs when they detect the mobile device is using wifi or a 4G/LTE network.

More modern codecs support a variable bit rate that can be changed automatically during a call to adapt to poor network conditions. The Opus codec used for WebRTC has this capability.

Recommendations

Enable as many codecs as possible to maximize the chance of connection. This reduces the number of calls where the endpoints fail to find a common set of codecs.

Disable those codecs that won't possibly work given the available bandwidth. For example, in a remote location with a 128kbit DSL broadband connection it is usually necessary to disable 64kbit codecs like G.711 and G.722 and use codecs that have lower audio quality such as Opus, GSM and G.729.

However, in a location with good bandwidth, don't disable the low-bit rate/low-quality codecs. They will still be needed when calling a user who doesn't have great bandwidth.

Order the remaining codecs based on the quality, with the best quality codec first. G.711 is typically present for compatibility but other codecs like G.722 offer better quality for the same amount of bandwidth, so rate G.722 ahead of G.711 and G.722 will be used whenever possible.

Example 1: a mobile app: enable, in order of priority starting with the most preferred: Opus, Silk and GSM codecs.

Example 2: a soft PBX in an office accessed by local users and mobile users: enable, in order of priority starting with the most preferred: Opus, G.722, G.711, Silk, iLBC, GSM, G.729.

Media stream encryption compatibility

RTC voice and video (media) streams are almost always transmitted using Real Time Protocol (RTP).

Encrypting RTP streams is a popular requirement. Some organizations are subject to regulations requiring encryption. In other cases, there is a commercial imperative to use encryption. Just as HTTP sessions can be encrypted with HTTPS, RTP streams can be encrypted using SRTP. SRTP is optimized for the real-time nature of the media stream and it permits packet loss. However, thanks to the way that SRTP has involved, just looking for the encryption setting and turning it on may lead to a big loss in compatability with other users if the products you are using don't support the right combination of encryption protocols. We consider the issues here and then provide some recommendations.

While SRTP itself is relatively standard, there are several different ways to exchange keys for an SRTP session. If the two endpoints trying to setup a call are not trying to use the same method for key exchange, or if one of them hasn't enabled encryption at all, *the call will not connect*. To meet our goal of maximizing connectivity, these mismatches must be avoided.

The original method of key exchange for SRTP involves exchanging SDES keys through the signalling channel, which may be a SIP or XMPP connection. Most of the original products supporting this standard take an all-or-nothing approach: if encryption is disabled, they will only talk to other endpoints without encryption and if encryption is enabled, they will only talk to other endpoints capable of the same key exchange protocol.

One disadvantage of sending the keys through the signalling channel is that the operator of the SIP proxy can easily observe the keys and use them to decrypt the RTP streams. Another risk is that the end-user has no way to know if a man-in-the-middle has swapped the keys, decrypting the streams, recording or modifying them and then sending them on to the other endpoint using a different key. Two alternative solutions to these problems have emerged.

Phil Zimmerman, the legendary creator of PGP encryption, created the ZRTP key exchange protocol. When the endpoints support ZRTP, they typically send signalling messages (SDP) specifying that regular, unencrypted RTP will be used for the call. When the call is answered, each endpoint tries to send some special ZRTP "hello" packets to the peer on the port normally used for the RTP. At this stage, the phones will indicate to the users that they are in a call without encryption. If the endpoints both support ZRTP then they recognize the "hello" packets from the peer and they perform a key exchange using the Diffie-Hellman algorithm. Once the key exchange is completed, the interface on the phone changes somehow to advise the user that the call is now encrypted. Due to the nature of the Diffie-Hellman algorithm and the verification of the algorithm by a short authentication string (SAS) that the users read to each other, the keys can not be observed or substituted by any man-in-the-middle.

Nonetheless, the use of the SAS may appear slightly geeky and it is only valid if you know the other person personally and can *recognize their voice* when they are reading the SAS to you. If you are calling an organization, such as the call center at the bank or a Government department, the person you are speaking to is likely to be a complete stranger, you will not have be familiar with the sound of their voice when they read the SAS and so you will not be able to rely on this algorithm.

The DTLS-SRTP standard provides another alternative, although on its own, it does not provide certainty that there is no man-in-the-middle. DTLS-SRTP provides a way to use DTLS key agreement before starting SRTP media streams. To provide security against a man-in-the-middle, DTLS-SRTP can be combined with another mechanism for the exchange of key fingerprints. For example, RFC 5763 specifies a mechanism for exchanging tamper-proof key fingerprints in SDP using SIP Identity (RFC 4474).

DTLS-SRTP is the mechanism that has been chosen for the WebRTC standard and it is widely implemented in web browsers. It is also supported by the Asterisk and FreeSWITCH projects and some softphones including Jitsi.

ZRTP is also supported in a number of products, including Asterisk, FreeSWITCH and the Jitsi softphone. ZRTP is not currently supported in WebRTC browsers although it is preferred by products that focus on the privacy of personal communication between people who know each other, such as the Lumicall app.

Supporting multiple schemes

For encryption to be effective, users must know when it is working. For this reason, many products started out with a simple all-or-nothing approach. If the encryption setting is enabled, the product will only permit calls to and from peers using them same encryption settings. Users could then conclude that any call that connects successfully is encrypted correctly.

Furthermore, the way that an SDP offer/answer exchange is designed, a media descriptor either has crypto attributes or it doesn't. There is no simple way for an endpoint to insert the attributes in the SDP and hint that they are optional. If they are present, the peer must act as if they are mandatory and reject the call if it is not capable of using encryption.

Some phones have an option to send two media descriptors in SDP, one of them with crypto attributes and the other without. However, some other phones don't understand this type of SDP and reject the call completely, so it is not a reliable strategy.

Some phones have an option to try the call with encryption enabled and if it is rejected, automatically try again without encryption. This strategy is not glamorous but it has wider compatibility.

Recommendations for maximizing connectivity

Generally, SDES SRTP should be avoided and should not enabled except for very specific cases. Do not enable SDES SRTP for arbitrary calls across the Internet as a means to improve compatability: the risks outweigh the benefits. The cases where you may use SDES SRTP include situations where you have IP phones that don't support any other form of encryption and connections to SIP trunking providers who don't support any other form of encryption. In these special cases, ensure that the SDES SRTP calls are only possible for the specific IP addresses or SIP accounts that you designate. Note that when you configure a connection profile in the PBX to use this encryption mode with certain peers, it may not be able to use the same connection profile for any other peers who don't use SDES SRTP. This is the all-or-nothing scenario.

When making calls, do not try to use the approach that involves sending multiple media descriptors, one with crypto attributes and the other without. For all other calls, use software that tries to make the call using DTLS-SRTP and if that fails retries the call using ZRTP. If encryption is not essential for you, allow calls to proceed even when ZRTP does not secure a connection.

When receiving calls, be willing to accept calls using either DTLS-SRTP or ZRTP and possibly without any encryption at all, depending upon your requirements. A SIP proxy can inspect the SDP to determine which type of encryption is attempted and route the call appropriately. For example, depending upon which attributes are present, the SIP proxy could route the call to an Asterisk `sip.conf` profile with DTLS-SRTP support or to another profile with ZRTP support. Alternatively, you may use a softphone that is capable of recognizing and accepting either type of encryption.

Throughout this guide, we present further details about how to achieve all of the above with the products described herein.

Recommendations for security

The recommendations in the previous section will help optimize connectivity, by enabling more calls to connect successfully. Additional steps are required to ensure you fully benefit from the security that encryption can provide, these are described here.

If you are enabling and relying on DTLS-SRTP with SIP, make sure you also use SIP Identity (RFC 4474) and use SIP Identity to authenticate the key fingerprints (RFC 5763).

If you pass calls through intermediate network components such as Session Border Controllers or a PBX where the media streams are decrypted and re-encrypted, you need to think carefully about the impact on authentication. For example, you may configure the PBX to enforce identity verification and then tell users that all calls that have come through the PBX can be trusted.

If using phones or softphones that are configured to work with or without encryption (this is referred to as *opportunistic encryption*), it is very desirable for them to give the user an indication about whether each call is encrypted. Otherwise, the user should be told to assume that calls are not encrypted.

Use ICE and a TURN server

A TURN server provides a standard way to relay media on behalf of users who are stuck on a NAT network. The Interactive Connectivity Establishment (ICE) protocol uses the TURN server to help explore network topology and give immediate feedback if the call is not possible, eliminating the menace of ghost calls.

Several TURN servers are now available in convenient Linux packages, see Chapter 10, *ICE/STUN/TURN server installation* for details about selecting and installing one.

Use the TLS transport for SIP signalling

When SIP messages are sent over UDP, there are several things that go wrong. The first problem is that large SIP messages can be fragmented by the IP stack and some fragments are not delivered.

When ICE is used, the SIP message contains a larger SDP body to encapsulate the ICE candidates. When a softphone attempts a video call, the combination of the video and audio descriptors further enlarges the SDP. More and more frequently in modern RTC deployments, SIP messages sent over UDP exceed the maximum transmission unit and are subject to fragmentation. IP packet fragments are not always routed correctly by other intermediate network components. This was not a problem in the early days of SIP when the vast majority of devices only supported a limited number of audio codecs and overall packet sizes were well under one kilobyte.

A more obscure issue is the presence of routers in homes and small offices that claim to have *SIP helper* capabilities. These routers try to modify the SIP messages to help them through NAT. In reality, the modifications made by the router can clash with the ICE protocol or other NAT discovery techniques used by the phone or the server.

Sending all the SIP messages over a TLS connection eliminates all of these problems. While there is slightly more effort involved to create a certificate for the server, it saves an enormous amount of ongoing support effort.

See Chapter 9, *TLS certificate creation* for details about creating the TLS certificates for SIP and XMPP.

Getting through firewalls

When a user is in a developed country, at their home or using a mobile Internet connection, they may be behind a NAT router but the firewall on these devices is usually very permissive for connections initiated by the user. In the vast majority of cases, the user will be able to initiate outbound TCP connections to any destination (such as a SIP, XMPP or WebSocket server on any arbitrary port number) and will be able to send and receive UDP.

For some NAT routers, the UDP flows will only work reliably when the peer is using a public IP address. This problem is automatically detected by ICE connectivity checks and it is resolved by sending the UDP packets through the TURN server.

For users with more repressive Internet providers, in some less sophisticated wifi hotspots and in some corporate networks there are more aggressive firewall policies. With a little care, the RTC deployment can be designed to work reliably in many of these environments too.

The first issue is the signalling connection, whether it is SIP, XMPP or WebSockets. More restricted corporate networks block outgoing TCP connections, except for those on port 80 or 443, which they redirect to a transparent proxy.

As a consequence of this TCP blocking, it should be anticipated that the user's softphone may need to use the HTTP proxy for all RTC traffic, including media streaming and signalling.

HTTP Proxy servers have a number of issues. Older proxy servers do not understand the WebSocket protocol and newer proxy servers are not always configured to allow the WebSocket protocol by default. To avoid these problems, it is recommended that the WebSocket connection should always use TLS (WebRTC clients uses a `wss://` URL instead of a `ws://` URL) and the port 443 only. Many HTTP proxy servers are correctly configured to allow the web browser to use the HTTP `CONNECT` method to initiate pass-through connections to `https://` URLs using the default port, 443. Sometimes, however, the HTTP proxy does not allow any port other than 443. Thanks to the encryption provided by TLS, the proxy server can not observe whether the HTTP `CONNECT` method is being used to reach a web server, a SIP server, a WebSocket server or even something else such as an `ssh` server. Therefore, all services, including SIP over TLS, XMPP client connectivity (`c2s`), TURN over TLS and WebSockets, should listen on port 443 so that any of them can be reached by a user stuck behind a HTTP proxy.

The next issue is the transmission of UDP. In some networks, users simply can not exchange any UDP packets with external hosts. The only resolution to this issue is to tunnel the UDP packets through TCP. Fortunately, the TURN server can also assist, as the TURN specification includes support for tunneling packets through a TLS connection. To maximize the chance of success, it is recommended that the TURN server is also configured to listen on port 443 so that the connection to the TURN server will be able to pass through HTTP proxy servers using the HTTP `CONNECT` method.

This strategy often requires several different processes (the standard SIP server, the XMPP `c2s` service, the webserver hosting a WebRTC phone, the WebSocket server and the TURN server) to all listen on the same port, 443. For multiple processes to use the same port, it is necessary to either have a different public IP address for each process or to use a solution for port multiplexing, such as the *sslh* daemon.

With these strategies, connectivity will be possible for the vast majority of Internet users, whether at home, at the office or on the road.

Chapter 5. DNS setup

The DNS records to be created are detailed in Table 5.1, “DNS records for the example”

Table 5.1. DNS records for the example

Record Name	Type	Value
`server1`	`A`	`198.51.100.19`
`server1`	`AAAA`	`2001:DB8:1000:2000::19`
`turn-server`	`A`	`198.51.100.19`
`turn-server`	`AAAA`	`2001:DB8:1000:2000::19`
`sip-proxy`	`A`	`198.51.100.19`
`sip-proxy`	`AAAA`	`2001:DB8:1000:2000::19`
`xmpp-gw`	`A`	`198.51.100.19`
`xmpp-gw`	`AAAA`	`2001:DB8:1000:2000::19`
`_stun._udp`	`SRV`	`0 1 3478 turn-server.example.org.`
`_turn._udp`	`SRV`	`0 1 3478 turn-server.example.org.`
`_sips._tcp`	`SRV`	`0 1 5061 sip-proxy.example.org.`
`_xmpp-client._tcp`	`SRV`	`5 0 5222 xmpp-gw.example.org.`
`_xmpp-server._tcp`	`SRV`	`5 0 5269 xmpp-gw.example.org.`
`@`	`NAPTR`	`10 0 "s" "SIPS+D2T" "" _sips._tcp.example.org.`
`@`	`NAPTR`	`10 0 "s" RELAY:turn.udp "" _turn._udp.example.org.`

Using non-standard ports

RTC makes use of DNS SRV records for load-balancing and failover. A key feature of the SRV record is that the TCP or UDP port number is specified in the record. Table 5.1, “DNS records for the example” demonstrates the use of standard port numbers for SIP, TURN and XMPP.

If users are connecting to the service from arbitrary locations, including public wi-fi hotspots, hotels and the offices of other companies, they will almost certainly encounter firewalls that only allow traffic to pass on a limited range of port numbers or through HTTP proxy servers.

For this reason, it is common to operate RTC services on port 443 instead of the normal port numbers. Two or more processes can't listen on the same port number on the same IP address. When all the RTC processes have to use port 443, it is necessary to have a different IP address for each process. Table 5.2, “Protocols using port 443” gives a summary of the ports to change.

Table 5.2. Protocols using port 443

Protocol	Default port	Non-standard port
STUN / TURN over TLS	5349	443
SIP over TLS	5061	443
XMPP client	5222	443

Sample DNS zone file

See Example 5.1, “ISC Bind zone file entries” for an example of how to write the entries for the zone file. For the purposes of the example, this file would be `/etc/bind/db.example.org` on the nameserver host.

Example 5.1. ISC Bind zone file entries

```
; the server where everything will run
server1            IN     A      198.51.100.19
server1            IN     AAAA   2001:DB8:1000:2000::19

; Use different names for each service.
; Don't use CNAMEs, the SRV records (further down)
; can't point to CNAME records.
turn-server        IN     A      198.51.100.19
turn-server        IN     AAAA   2001:DB8:1000:2000::19
sip-proxy          IN     A      198.51.100.19
sip-proxy          IN     AAAA   2001:DB8:1000:2000::19
xmpp-gw            IN     A      198.51.100.19
xmpp-gw            IN     AAAA   2001:DB8:1000:2000::19


; DNS SRV for STUN / TURN
_stun._udp  IN SRV    0 1 3478 turn-server.example.org.
_turn._udp  IN SRV    0 1 3478 turn-server.example.org.

; DNS SRV and NAPTR records for SIP
_sips._tcp  IN SRV    0 1 5061 sip-proxy.example.org.
@           IN NAPTR  10 0 "s" "SIPS+D2T" "" _sips._tcp.example.org.
@           IN NAPTR  10 0 "s" RELAY:turn.udp "" _turn._udp.example.org.

; DNS SRV records for XMPP Server and Client modes:
_xmpp-client._tcp  IN     SRV    5 0 5222 xmpp-gw.example.org.
_xmpp-server._tcp  IN     SRV    5 0 5269 xmpp-gw.example.org.
```

Testing the DNS settings

Use the `dig` command to test, as demonstrated in Example 5.2, “Inspecting DNS entries with `dig`”

Example 5.2. Inspecting DNS entries with `dig`

```
$ dig -t naptr +short example.org
10 0 "s" "SIPS+D2T" "" _sips._tcp.example.org.

$ dig -t srv +short _sips._tcp.example.org.
0 1 5061 sip-proxy.example.org.
```

Chapter 6. Firewall rules

Overview of firewall ports

Table 6.1, “Firewall rules summary” and Table 6.2, “Firewall rules summary (IPv6)” list each firewall rule that is required for the test addresses described in the section called “Example network used in the documentation”.

Table 6.1. Firewall rules summary

Purpose	IP Protocol	Source Addr	Source Port	Dest Addr	Dest Port
RTP media (both SIP and XMPP, audio and video)	UDP	Any	Any	`198.51.100.19`	49152 to 65535
TURN session control	UDP	Any	Any	`198.51.100.19`, `198.51.100.20`	3478
STUN NAT discovery (RFC 3489)	UDP	Any	Any	`198.51.100.19`, `198.51.100.20`	3479
SIP signalling	TCP	Any	Any	`198.51.100.19`	5061
SIP over WebSocket (TLS)	TCP	Any	Any	`198.51.100.19`	443
XMPP Server signalling	TCP	Any	Any	`198.51.100.19`	5222
XMPP Client signalling	TCP	Any	Any	`198.51.100.19`	5269

Table 6.2. Firewall rules summary (IPv6)

Purpose	IP Protocol	Source Addr	Source Port	Dest Addr	Dest Port
RTP media (both SIP and XMPP, audio and video)	UDP	Any	Any	`2001:DB8:1000:2000::19`	49152 to 65535
TURN session control	UDP	Any	Any	`2001:DB8:1000:2000::19`	3478
SIP signalling	TCP	Any	Any	`2001:DB8:1000:2000::19`	5061
SIP over WebSocket (TLS)	TCP	Any	Any	`2001:DB8:1000:2000::19`	443
XMPP Server signalling	TCP	Any	Any	`2001:DB8:1000:2000::19`	5222
XMPP Client signalling	TCP	Any	Any	`2001:DB8:1000:2000::19`	5269

NAT considerations

Many networks use NAT to minimize cost, conserve public IP addresses and to avoid direct routing from the public Internet. RTC applications can work in a NAT environment, however, there are some points to be aware of.

One common technique used for web servers involves hosting the public IP address on the firewall and creating a port forwarding rule redirecting all incoming connections to the internal IP address of a web server. This approach works for some types of services, such as HTTP, but it does not work for all types of RTC traffic. In particular, it is essential that the TURN server process runs on a host with two public IP addresses. A SIP server may work with port forwarding, but care needs to be taken to ensure the record-route URI matches the external IP address. Using SIP over TLS and SIP over WebSockets with port forwarding is more likely to work than trying to port-forward SIP over UDP traffic.

The TURN server does not need to have an IP address on the private network but it does need to be routable from the private network. The TURN server could be hosted in a DMZ or even using an external hosting provider.

If you choose to operate a SIP Session Border Controller (SBC), it will probably need to have both a public IP address and a private IP address.

Setup with `iptables` on Linux

Example 6.1, "Firewall setup with `iptables`" provides a basic example for Linux firewalls using `iptables`. If using a firewall framework like *Shorewall* then please consult the relevant documentation to open the same ports.

Example 6.1. Firewall setup with `iptables`

```
iptables -I INPUT -p udp -d 198.51.100.19 --dport 3478 -j ACCEPT
iptables -I INPUT -p udp -d 198.51.100.20 --dport 3478 -j ACCEPT
iptables -I INPUT -p udp -d 198.51.100.19 \
        --dport 49152:65535 -j ACCEPT

iptables -A INPUT -p tcp -d 198.51.100.19 --dport 5061 -j ACCEPT

iptables -A INPUT -p tcp -d 198.51.100.19 --dport 5222 -j ACCEPT
iptables -A INPUT -p tcp -d 198.51.100.19 --dport 5269 -j ACCEPT

ip6tables -I INPUT -p udp -d 2001:DB8:1000:2000::19 \
        --dport 3478 -j ACCEPT
ip6tables -I INPUT -p udp -d 2001:DB8:1000:2000::19 \
        --dport 49152:65535 -j ACCEPT

ip6tables -A INPUT -p tcp -d 2001:DB8:1000:2000::19 \
        --dport 5061 -j ACCEPT

ip6tables -A INPUT -p tcp -d 2001:DB8:1000:2000::19 \
        --dport 5222 -j ACCEPT
ip6tables -A INPUT -p tcp -d 2001:DB8:1000:2000::19 \
        --dport 5269 -j ACCEPT
```

It is highly recommended that the firewall rules for RTP packets are placed at the beginning of the chain, as these packets are time sensitive and over 99% of the RTC traffic is carried in the RTP packets. Putting them lower in the chain will mean that the CPU does more work evaluating each packet before it finds the matching rule. That would lead to wasted CPU cycles and potential latency or congestion issues for all real-time applications on the server.

When you deploy additional RTC applications (such as Asterisk or FreeSWITCH) behind the firewall, you may want to allow RTP traffic to travel directly to those servers too while only allowing the SIP traffic to go through the SIP proxy.

Chapter 7. User and credential storage

Most of the products described in this document, including SIP proxy servers, XMPP servers, TURN servers and soft PBXes offer a range of choices for maintaining a user database and storing user credentials.

Credentials

Personal account names or extension numbers

An issue that arises early in many discussions about this topic is whether to use email addresses of the form `username@example.org` or to use extension numbers as the account identifiers.

In most cases, users still need to be able to be contactable using either type of address, but the question remains: should their phone login to the system using a named username or using an extension number?

Extension numbers are the standard in many traditional phone systems and some people have simply tried to replicate this model when moving to IP-based RTC.

There are three big reasons why numbers are popular: numbers can be dialed from anywhere else in the PSTN, numbers are easier to dial for people who are not calling from a computer (phones only have 12 buttons) and there is not always a one-to-one mapping between people and phones. One additional feature of numbers is that they are slightly less personal, somebody does not know exactly who will answer when they call a landline in a house shared by a large family and when somebody leaves a job, their phone number can be assigned to their successor.

Mobile telephones and smartphones in particular have dramatically reduced the significance of these factors that encouraged the continued use of numbers. For example, the inconvenience of manually dialing a SIP or XMPP address is not such a big factor because people are more likely to have these details in their address books.

Phone numbers also have disadvantages. One of these is the dependency on phone companies, the ITU and government bureaucracies who administer the numbering system. Users can be forced to change their phone number when changing provider. Dialing codes change, making it necessary to update old records in an address book. Each group of numbers is tightly bound to the infrastructure of a specific the telephone exchange, leading to inflexibility when relocating or when a local service outage occurs.

Recommendation

Use alphabetic usernames rather than extension numbers internally. SIP, XMPP and email each support a slightly different set of characters in usernames so only use the set of characters supported for all protocols. Wherever a device is being used with named accounts, such as a desktop PC where users login and have access to their own company email account, or a smartphone that is only used by one person and has been configured to access a named email account, provision the same personalized SIP/XMPP address on that device. Where a device is shared, such as a conference room phone, create a named account for that purpose.

It is generally helpful if the same username is used for login, email address, SIP and XMPP addresses. If the login names are not the same as email addresses, use the email addresses as the SIP and XMPP addresses.

To give users the convenience of dialing extension numbers from regular phones, create mappings from the extension numbers to the usernames.

This strategy offers significantly more potential for the future as more and more services will rely on usernames and fewer services will place emphasis on phone numbers.

When using this strategy, it is necessary to implement mappings from phone numbers to usernames as part of the *ingress* processing and for calls that go out to the PSTN over SIP or ISDN trunks, it is necessary to implement the reverse mapping, ensuring that the caller ID of each outgoing call is personalized based on the internal username.

Password encryption

It is not strictly necessary to use passwords for SIP, XMPP and TURN, you can use certificate authentication instead. Nonetheless, many people find that some of their users can only support password authentication or they don't want the complexity of managing public key infrastructure (PKI).

XMPP transmits passwords in cleartext, so all XMPP connections should be secured with TLS to avoid eavesdropping. The benefit of this approach is that XMPP users can be authenticated against any type of pre-hashed passwords or using a method such as LDAP bind to verify the supplied credential.

SIP and TURN use a DIGEST algorithm very similar to HTTP DIGEST. The DIGEST algorithm requires the server to have either an unencrypted copy of the password, a password encrypted with the HA1 algorithm or a service (such as RADIUS) that can perform delegated DIGEST authentication.

If unencrypted passwords are available, then the SIP and TURN servers can use them to construct the HA1 hash value or you can precompute the HA1 values and store them in a database.

Warning

HA1 hash values should be considered as sensitive as unencrypted passwords. Even though the plaintext password can't be recovered in a simple manner, anybody in possession of the HA1 value is able to use it to construct a response to a HTTP or SIP DIGEST challenge, thereby impersonating the user who owns that password. Therefore, do not keep HA1 hash values in world-readable configuration files or publicly accessible in LDAP.

HA1 in detail

The HA1 hash includes the username, the password and the realm. A HA1 hash can be easily constructed at the UNIX command line as demonstrated in Example 7.1, “Computing HA1”.

Example 7.1. Computing HA1

```
$ echo -n "alice:example.org:secret" | md5sum
543e1aec5d3614f03141652d6ada51b2  -
$ echo -n "alice@example.org:example.org:secret" | md5sum
1441a999b257bcd0cf5166930039876a  -
```

In both examples, the realm is `example.org`.

In the first case, the user authenticates using the username `alice`. In the second case, the user authenticates using the username `alice@example.org`. This second permutation is referred to by some products as *HA1B*. Some people prefer to use the full *user@domain* syntax to support virtual hosting with a single *realm* value on a single SIP proxy or TURN server.

Databases

RADIUS

IETF `draft-sterman-aaa-sip-04` describes a mechanism for RADIUS servers to participate in a SIP DIGEST challenge/response without the SIP proxy having a copy of the password or HA1 value at all. This is implemented by the *FreeRADIUS* project in the module `rlm_digest` [http://wiki.freeradius.org/modules/Rlm_digest] and supported by all the major SIP proxy servers.

At the time of writing, work is in progress to enable RADIUS to participate in TURN authentication in the same way.

LDAP

If LDAP is in use, you may wish to consider storing the HA1 values in the LDAP directory. Each time a user is created or a user changes their password, the LDAP server will need to update the HA1 hash as well as updating any other copies of the password hashed with other algorithms.

For example, the *OpenLDAP* server allows such logic to be implemented in an overlay, this is already demonstrated in the *smbk5pwd* module for hashing copies of the user's password in various algorithms used by Windows.

Due to the sensitive nature of the HA1 values, they should be stored in an attribute that is not readable to any other user or anonymous access. Example 7.2, “*OpenLDAP* ACL for protecting `ha1Password`” demonstrates how to protect the `ha1Password` so it can only be read by a user `cn=sip-proxy,dc=example,dc=org`.

Example 7.2. *OpenLDAP* ACL for protecting `ha1Password`

```
    access to attr=ha1Password
        by self =xw
        by dn="cn=sip-proxy,dc=example,dc=org" read
        by anonymous auth
        by * none
```

LDAP can also be used to assist in routing as described in Chapter 15, *Adding ENUM to DNS*.

SQL databases

Many RTC products have some capability to interact with an SQL database to obtain user credentials, configuration settings and routing information. Example 7.3, “SQL table for repro users” demonstrates a typical schema.

Example 7.3. SQL table for repro users

```
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  username VARCHAR(64) NOT NULL,
  domain VARCHAR(253),
  realm VARCHAR(253),
  passwordHash VARCHAR(32),
  passwordHashAlt VARCHAR(32),
  name VARCHAR(256),
  email VARCHAR(256),
  forwardAddress VARCHAR(256)
);
```

Each product has a different schema, however, it is possible to create a user list table that aggregates all the columns required to satisfy multiple processes and then create SQL views to present the data with the column names required by the individual applications. For example, a single user list table can be created for use by Asterisk (the `sippeers` table) and used by repro (using a view instead of a real `users` table) as demonstrated in Example 7.4, “SQL view presenting Asterisk users to repro”. repro simply doesn't require many of the columns used by Asterisk. Notice that Asterisk's `sippeers` table doesn't contain a domain or realm column for each user, these are stored elsewhere in the `sip.conf` file, so for repro, they are specified as constant values in the `SELECT` query.

Example 7.4. SQL view presenting Asterisk users to repro

```
CREATE VIEW users AS
  SELECT
    id,
    name AS username,
    'my_domain' AS domain,
    'my_realm' AS realm,
    md5secret AS passwordHash,
    NULL As passwordHashAlt,
    NULL AS name,
    NULL AS email,
    NULL AS forwardAddress
  FROM sippeers;
```

Setting up PostgreSQL for SIP users

The packages for the *repro* SIP proxy include SQL schema files for creating tables. Example 7.5, “Install *PostgreSQL* on Debian or Ubuntu” demonstrates how to install the *PostgreSQL* server package. Example 7.6, “Configure PostgreSQL and load schema” demonstrates how to use the `createdb` command to create a database called `repro` and use the `psql` command to log in as the *DBA* and create a user called `repro` for the SIP proxy. The final `psql` command uses the schema file to create the tables.

Example 7.5. Install *PostgreSQL* on Debian or Ubuntu

```
$ sudo apt-get install postgresql postgresql-client
```

Example 7.6. Configure PostgreSQL and load schema

```
$ sudo su - postgres
postgres$ createdb repro
postgres$ psql
postgres=# create user repro password 'abc';
postgres=# \q
postgres$ exit
$ su vi /etc/postgresql/9.4/main/pg_hba.conf
$ su systemctl restart postgresql
$ psql -U repro -W repro < /usr/share/doc/repro/create_postgresql_reprodb.sql
```

Product-specific file formats

Each product also supports some native file formats. For example, *repro* can store user data in *Berkeley DB* files while *Asterisk* can store users in the `sip.conf` text file. The *Prosody* XMPP server can store its data in JSON files. In some cases, the files are maintained by the administrator using a text editor and in other cases they are updated at runtime by the application.

To eliminate the risk of runtime dependencies on databases, it is relatively straightforward to create a script that periodically extracts user data from a database and creates files for the relevant processes to consume. If one of the processes has to start up or continue operating during a database outage, it will be able to do so using the last copy of the file.

Conclusion

A guide like this can't proscribe the correct solution for every scenario. This chapter simply aims to raise awareness of all the options for storing usernames and credentials and making them accessible to RTC processes. System administrators and developers will need to consider the infrastructure that is already present when deciding which of these options are most relevant for a given site.

Chapter 8. Server setup

The packages are available in a convenient format in popular Linux operating systems including Debian [http://www.debian.org/distrib/], Ubuntu [http://www.ubuntu.com] and Fedora [http://fedoraproject.org/get-fedora-all]. the section called “Choice of operating system” discusses the choice of operating system.

If using *Debian 8 (jessie)* you must use the reSIProcate 1.10.x packages from jessie-backports [http://backports.debian.org/Instructions/].

Users of Red Hat Enterprise Linux (RHEL) and CentOS can easily build the reSIProcate package using `rpmbuild`. All required dependencies are available in the operating system or the EPEL collection.

It is strongly recommended that you either use *1.9.7-4* or later. Previous versions available in older distributions do not have the most recent fixes for OpenSSL `SSLv23_method` issues. For any new installations, it is recommended to start with version *1.10.0* or later as it offers several more improvements in features and interoperability.

Install the operating system using the normal process, set up an IP address on the machine and make sure network connectivity is working.

If you are not familiar with server installation, a useful resource is the Debian Administrator's Handbook [http://debian-handbook.info].

Debian/Ubuntu servers

To set up both of the IP addresses on the same box (as required by the ICE/STUN/TURN protocol), modify the `/etc/network/interfaces` file as demonstrated in Example 8.1, “Adding IP addresses in `/etc/network/interfaces`”.

Example 8.1. Adding IP addresses in `/etc/network/interfaces`

```
allow-hotplug eth0
iface eth0 inet static
        address 198.51.100.19
        netmask 255.255.255.0
        network 198.51.100.0
        broadcast 198.51.100.255
        gateway 198.51.100.1
        up ip addr add dev eth0 198.51.100.20/24 scope global

iface eth0 inet6 static
        address 2001:DB8:1000:2000::19
        netmask 64
        gateway 2001:DB8:1000:2000::1
        dns-nameservers 2001:DB8:1000:2000::5
        dns-search example.org
```

Notice the line using the `ip addr add` command to add the additional IP address.

Fedora, RHEL and CentOS servers

To set up both of the IP addresses on the same box (as required by the ICE/STUN/TURN protocol), see the documentation about alias and clone files [http://docs.fedoraproject.org/en-US/Fedora/17/html/System_Administrators_Guide/s2-networkscripts-interfaces-alias.html] for network interfaces

Chapter 9. TLS certificate creation

Certificates are an essential security mechanism for most federated and distributed technologies on the Internet.

Certificates may be referred to as X.509 certificates, SSL certificates or TLS certificates. For most purposes, these terms all refer to the same thing and the term TLS certificate is used throughout this documentation.

The prices of TLS certificates vary significantly. It is not necessarily useful to purchase the most expensive one.

The free TLS certificates from the Let's Encrypt Project [https://letsencrypt.org/], which is supported by the EFF and Linux Foundation, are a good choice for the vast majority of RTC projects, including WebRTC. Let's Encrypt is not just a new Certificate Authority, they also promote the use of an automated tool for the acquisition and renewal of certificates. This dramatically reduces the amount of manual effort involved in using certificates, especially for people who host multiple sites and domains. That said, the initial version of the Let's Encrypt tool has been designed for use with web servers and some manual tweaking is required to use it with SIP, XMPP, WebSocket and TURN servers. Early versions of the tool also failed to operate correctly on some servers with IPv6 addresses and some Apache configurations, although most of these issues were resolved by mid-2016.

You do need to make sure that the certificate issued by the Certificate Authority (CA) includes both the *TLS client* and *TLS server* Extended Key Usage (EKU) extensions, some only include the latter. The free certificates from *StartSSL/StartCom* do not have the *TLS client* extension and can't be used. The Gandi.net SSL Standard certificate [https://www.gandi.net/ssl/standard#single] which costs about $16 (free with a domain registration or transfer) is known to be suitable.

If you are using some older IP desk phones, the phones may not have support for the Let's Encrypt root certificate in their firmware. If this is the case, you may need to update the firmware, obtain a newer model phone or use certificates from a more established Certificate Authority. For example, some older Polycom phones do not work with Let's Encrypt but they work fine with the low cost Gandi.net certificates.

Certificate Common Name

Certificates confirm the identity of a service. The identity is specified by the *Common Name* (CN) and in some cases the `subjectAltName` embedded in the certificate.

Some vendors refer to `subjectAltName` certificates as SAN certificates. This acronym is more commonly used for Storage Area Network and can cause confusion.

Early versions of the federated XMPP specification proposed a custom OID, `xmppAddr`, rather than using `subjectAltName`. This practice was not widely supported by certificate authorities. Furthermore, it meant that such certificates could not be used for purposes other than XMPP, such as SMTP email or SIP. The XMPP specification has since been relaxed and it is now possible to use a single certificate on a server for SIP, XMPP, SMTP and other purposes.

Many web sites use a name such as `www.example.org` and include the `www` prefix in the CN in their certificate. When purchasing a certificate for SIP and XMPP, it is important to ensure that the certificate contains a CN or `subjectAltName` that specifies the domain alone. For the `example.org` domain, the certificate should include `CN=example.org` or `subjectAltName=example.org` and not something like `CN=www.example.org`.

In a *wildcard certificate*, the CN will include an asterisk (`*`), for example, `CN=*.example.org`. This type of certificate can be used for the domain `example.org` and subdomains or hostnames such as `www.example.org` or `mail.example.org`.

Some organizations have *wildcard* certificates for all servers/subdomains in the organization. These are not always suitable for RTC purposes, in particular, RFC 5922 section 7.2 [https://tools.ietf.org/html/rfc5922#section-7.2] prohibits the use of wildcard certificates for SIP. Some SIP products offer the ability to override this restriction and use wildcard certificates anyway, however, this is not suitable for the public Internet as you can't be sure that other servers will have the same override enabled.

The correct domain needs to be specified when creating the *certificate signing request* (CSR) and should be confirmed by the CA in their web-based ordering form. If using the Let's Encrypt utility to obtain certificates, this part of the process is automated.

Install the OpenSSL utility

Make sure the OpenSSL package is available, it can be installed using the package manager as demonstrated in Example 9.1, “Installing `openssl` on Debian/Ubuntu” and Example 9.2, “Installing `openssl` on Fedora/RHEL/CentOS”.

Example 9.1. Installing `openssl` on Debian/Ubuntu

```
$ sudo apt-get install ssl-cert openssl
```

Example 9.2. Installing `openssl` on Fedora/RHEL/CentOS

```
$ sudo yum install openssl
```

Install the Let's Encrypt `certbot` utility

If you have decided to use Let's Encrypt certificates, it is necessary to install the `certbot` utility or an equivalent utility implementing the Automated Certificate Management Environment (ACME) [https://en.wikipedia.org/wiki/Automated_Certificate_Management_Environment]. Install `certbot` using the package manager as demonstrated in Example 9.3, “Installing `certbot` on Debian/Ubuntu” and Example 9.4, “Installing `certbot` on Fedora/RHEL/CentOS”.

Example 9.3. Installing `certbot` on Debian/Ubuntu

```
$ sudo apt-get install -t jessie-backports ssl-cert certbot
```

Example 9.4. Installing `certbot` on Fedora/RHEL/CentOS

```
$ sudo yum install certbot
```

Install a TLS certificate using Let's Encrypt (certbot)

If using a web server for exactly the same domain name as your RTC service, it is possible to use the `certbot` command for your web server to setup the certificate the first time.

If there is no HTTPS web server for the domain, it is possible to use the `certbot certonly` subcommand to request a certificate.

As the `certbot` utility is still quite new and evolving, it is recommend that you consult the `certbot` web site [https://certbot.eff.org/] for the most up-to-date detailed instructions.

After running `certbot`, the certificate files will be present at locations such as `/etc/letsencrypt/live/example.org/fullchain.pem` and the private key files will be present at locations such as `/etc/letsencrypt/live/example.org/privkey.pem`

Install a TLS certificate manually

If you have decided not to use Let's Encrypt and `certbot`, use these instructions to create your certificate(s) manually. Otherwise, this section can be skipped.

On each Linux platform, there are different locations for private key files and local server certificates. To simplify the examples, we define environment variables referring to them. See Example 9.5, “PKI directories (Debian/Ubuntu)” and Example 9.6, “PKI directories (Fedora/RHEL/CentOS)”.

On the server, create an *RSA key pair* and a *certificate signing request* (CSR) as demonstrated in Example 9.7, “Creating RSA key pair and CSR”.

Example 9.5. PKI directories (Debian/Ubuntu)

```
$ PKI_HOME=/etc/ssl
$ PRIVATE_KEY_DIR=${PKI_HOME}/private
$ CERT_DIR=${PKI_HOME}/public
$ CSR_DIR=${PKI_HOME}/csr
```

Example 9.6. PKI directories (Fedora/RHEL/CentOS)

```
$ PKI_HOME=/etc/pki/tls
$ PRIVATE_KEY_DIR=${PKI_HOME}/private
$ CERT_DIR=${PKI_HOME}/certs
$ CSR_DIR=${PKI_HOME}/csr
```

Example 9.7. Creating RSA key pair and CSR

```
$ MY_DOMAIN=example.org
$ sudo mkdir -p $PRIVATE_KEY_DIR
$ PRIVATE_KEY_PEM=${PRIVATE_KEY_DIR}/${MY_DOMAIN}-key.pem
$ CSR_PEM=${CSR_DIR}/${MY_DOMAIN}-csr.pem
$ sudo openssl genrsa -out ${PRIVATE_KEY_PEM} 2048
$ sudo chmod 0640 ${PRIVATE_KEY_PEM}
$ sudo chgrp ssl-cert ${PRIVATE_KEY_PEM}
$ sudo mkdir -p ${CSR_DIR} ${CERT_DIR}
$ sudo openssl req -new \
          -key ${PRIVATE_KEY_PEM} \
          -out ${CSR_PEM} \
          -subj "/CN=${MY_DOMAIN}"
$ sudo cat ${CSR_PEM}
```

Your CA will ask you to copy the CSR text and paste it into a form on their web site. The CA will now issue a certificate; it may be displayed in the browser or sent to you by email. Copy and paste it into the server after the `cat` command in Example 9.8, “Installing the certificate”, pressing `CTRL-D` or typing `EOF` to finish.

If the CA provides an *intermediate certificate*, you must also append it to the certificate file. The certificate file should contain the certificate for your domain, following by each intermediate certificate in order up to but not including the root.

Example 9.8. Installing the certificate

```
$ sudo cat > /etc/ssl/public/${MY_DOMAIN}.pem << EOF
-----BEGIN CERTIFICATE-----
MIIHWTCCBUGgAwIBAgIDCkGKMA0GCSqGSIb3DQEBCwUAMHkxEDAOBgNVBAoTB1Jv
b3QgQ0ExHjAcBgNVBAsTFWh0dHA6Ly93d3cuY2FjZXJ0Lm9yZzEiMCAGA1UEAxMZ
Q0EgQ2VydCBTaWduaW5nIEF1dGhvcml0eTEhMB8GCSqGSIb3DQEJARYSc3VwcG9y
```

```
.
.
.
d+pLncdBu8fA46A/5H2kjXPmEkvfoXNzczqA6NXLji/L6hOn1kGLrPo8idck9U60
4GGSt/M3mMS+lqO3ig==
-----END CERTIFICATE-----
EOF
```

The certificate is now ready for use by both the SIP and XMPP servers. It can also be used to secure a web server, SMTP server or any other application.

Chapter 10. ICE/STUN/TURN server installation

Choosing a TURN server

There are several TURN servers you can choose from. Any TURN server works for SIP, TURN, WebRTC and other protocols.

reTurnServer is the TURN server from the reSIProcate project. It is easy to set up using the packages, instructions are below.

CoTurn evolved from the *rfc5766-turn-server* project. See the CoTurn web site for instructions [https://code.google.com/p/coturn/] and then come back to this document to continue setting up your RTC environment.

TurnServer.org comes from the Jitsi [http://jitsi.org] team. See the TurnServer.org web site for instructions [http://www.turnserver.org] and then come back to this document to continue setting up your RTC environment.

restund is another option. See the `restund` web site [http://www.creytiv.com/restund.html] for details.

There are various factors to consider when choosing a TURN server.

Scalability: if you need to support thousands of users or more, you will want to test each of the servers for performance and evaluate the clustering capabilities.

Authentication: where do you store your user credentials? TURN servers use the same HA1 hashed passwords that HTTP DIGEST and SIP authentication uses so if you have such passwords in a database or LDAP server already you will want to evaluate which TURN servers can use that database or look at options for exporting the credentials into a file format for the TURN server.

Packaging: is the TURN server supported in a package on common Linux distributions? Most of those on the list above can be installed using official packages on Debian, Ubuntu and Fedora. Many people prefer to use packages so they don't have to spend time building from source code.

IPv6: do you need IPv6 support?

reTurnServer from reSIProcate

Installation

Install the package using the appropriate tool, as demonstrated in Example 10.1, “Installing `reTurnServer` on Debian/Ubuntu” and Example 10.2, “Install `reTurnServer` on Fedora/RHEL/CentOS”. If the package is not available for your platform, you may be able to build it using the instructions in Appendix B, *Building reSIProcate RPMs on RHEL and CentOS*.

Example 10.1. Installing `reTurnServer` on Debian/Ubuntu

```
$ sudo apt-get install resiprocate-turn-server
```

Example 10.2. Install `reTurnServer` on Fedora/RHEL/CentOS

```
$ sudo yum install resiprocate-turn-server
```

Configuration

Edit the configuration file, `/etc/reTurn/reTurnServer.config`, there are certain values that **must** be changed from the default values. These are demonstrated in Example 10.3, “`reTurnServer.config` entries”.

Example 10.3. `reTurnServer.config` entries

```
# your IP addresses go here:
TurnAddress = 198.51.100.19
TurnV6Address = 2001:DB8:1000:2000::19
AltStunAddress = 198.51.100.20
AltStunPort = 3479
# your domain goes here, it must match the value used
# to hash your passwords if they are already hashed
# using the HA1 algorithm:
AuthenticationRealm = example.org

UserDatabaseFile = /etc/reTurn/users.txt
UserDatabaseHashedPasswords = true
```

The host `server1` in this example MUST have two IP addresses, in the example, 198.51.100.19 and 198.51.100.20. This is essential for the ICE/STUN/TURN protocols.

Now (re)start the `reTurnServer` daemon to use the new settings as demonstrated in Example 10.4, “Restarting the `reTurnServer` daemon (`systemd`)”

Example 10.4. Restarting the `reTurnServer` daemon (`systemd`)

```
$ sudo systemctl restart resiprocate-turn-server
Restarting TURN relay: reTurnServer.
$
```

The TURN server should now be running and listening for client connections. You can verify it is running as demonstrated in Example 10.5, “Using `netstat` to verify `reTurnServer` is running”.

Example 10.5. Using `netstat` to verify `reTurnServer` is running

```
$ sudo netstat -nlp | grep reTurnServer
udp   0   0 198.51.100.19:3478      0.0.0.0:*              2460/reTurnServer
udp   0   0 198.51.100.20:3478      0.0.0.0:*              2460/reTurnServer
...
```

Check the system log for messages or run it in foreground mode on the console if it fails to start.

Provisioning users

The `reTurnServer` daemon expects to load a list of users and password hashes from a text file specified by the `UserDatabaseFile` parameter in `reTurnServer.config`.

Note that the order of the columns in this file is not the same as that used by `repro` and the `htdigest` utility.

The file can be generated by using a script to read values from a database table or LDAP directory.

The `reTurnServer` caches the file in memory when it starts. If the file is modified or regenerated while `reTurnServer` is running, send it the `HUP` signal to reload the file without restarting.

Synchronizing users from a PostgreSQL table

When the users are stored in a PostgreSQL table, such as the `users` table used by the `repro` daemon, the `psql-user-extract` script from reSIProcate can be used to maintain the `users.txt` file for *reTurnServer*.

The script is contained in a separate package or it can be downloaded directly from the source repository.

`psql-user-extract` can be invoked from `cron`, see Example 10.6, “`crontab` entry for `psql-user-extract`”.

Example 10.6. `crontab` entry for `psql-user-extract`

```
* * * * * /usr/lib/resiprocate/reTurnServer/psql-user-extract
```

`psql-user-extract` requires a configuration file specifying the database connection parameters, see Example 10.7, “Sample `/etc/reTurn/psql-user-extract.config`”.

Example 10.7. Sample `/etc/reTurn/psql-user-extract.config`

```
psql_conninfo = "dbname=repro user=repro host=localhost password=foobar"

# create this directory if it doesn't exist
dest_file = "/var/cache/reTurn/users.txt"

auth_user_alt = True
```

Testing the TURN server

As the TURN server speaks the STUN protocol, a simple way to test it is with a STUN client.

Example 10.8. Installing the `stun` client utility

```
$ sudo apt-get install stun
```

Example 10.9. Using the `stun` client utility

```
$ stun turn-server.example.org
STUN client version 0.96
Primary: Firewall
Return value is 0x00000b
```

Chapter 11. SIP proxy server installation

Choose your SIP proxy

Table 11.1. Comparison of SIP proxy servers

Feature	repro	Kamailio
Packages	Available in Debian, Ubuntu and Fedora	
Other	Either SIP proxy can be installed from source code on any platform	
Ease of installation	Single config file, federated mode is enabled by simple config settings	Flexible config file format, which is more like a scripting language and suitable for advanced customization. If a sample config from the Kamailio teams meets your needs, then it is very easy to install.
Module/plugin support	Extensions can be developed in C++ or Python	Modular plugin architecture with many plugins available
Database/user storage	Both support a common SQL table structure, so you can start with one and switch to the other, using either PostgreSQL or MySQL server.	
Management UI	HTML web interface enabled by default	Optional HTML UI available

Recommendation: If you are not sure, or can't find a suitable Kamailio configuration file for your needs, start with repro and you can change to Kamailio later if you find a reason to do so.

repro SIP proxy

Package installation

Install the package using the appropriate tool, as demonstrated in Example 11.1, “Installing `repro` on Debian/Ubuntu” and Example 11.2, “Install `repro` on Fedora/RHEL/CentOS”. If the package is not available for your platform, you may be able to build it using the instruction in Appendix B, *Building reSIProcate RPMs on RHEL and CentOS*.

Example 11.1. Installing `repro` on Debian/Ubuntu

```
$ sudo apt-get install repro
$ sudo addgroup repro ssl-cert
```

Example 11.2. Install `repro` on Fedora/RHEL/CentOS

```
$ sudo yum install resiprocate-repro
$ sudo addgroup repro ssl-cert
```

Configuration

The configuration file is `/etc/repro/repro.config`. Make essential changes to the configuration file, all other values can remain with default settings. Example 11.3, “Sample values for `repro.config`” demonstrates the main things you need to add or change from default values.

An important thing to note about the example is that we explicitly configure each transport. The `Transport1...` settings declare a TLS transport binding on a specific IP address and port. When one or more transports are defined in this way, the settings for global transport configuration, such as `IPAddress` and `UDPPort`, are completely ignored. For SIP to work reliably, binding to specific IP addresses is highly recommended. This ensures that outgoing TCP or TLS connections always use the correct source address and that the intended addresses are always included in `Via` headers.

The exact location of the `*.pem` files can be changed if necessary.

Example 11.3. Sample values for `repro.config`

```
# To trust email certificates as SIP client certificates:
TLSUseEmailAsSIP = true

# Transport1 will be for SIP over TLS connections
# We use port 5061 here but if you have clients connecting from
# locations with firewalls you could change this to listen on port 443
Transport1Interface = 198.51.100.19:5061
Transport1Type = TLS
Transport1TlsDomain = example.org
Transport1TlsClientVerification = Optional
Transport1RecordRouteUri = sip:example.org;transport=TLS
# Configuration for manually maintained TLS certificates:
Transport1TlsPrivateKey = /etc/ssl/private/example.org-key.pem
Transport1TlsCertificate = /etc/ssl/public/example.org.pem
# Configuration for certificates obtained using Let's Encrypt / certbot:
#Transport1TlsPrivateKey = /etc/letsencrypt/live/example.org/privkey.pem
#Transport1TlsCertificate = /etc/letsencrypt/live/example.org/fullchain.pem

# Transport2 is the IPv6 version of Transport1
Transport2Interface = 2001:DB8:1000:2000::19:5061
Transport2Type = TLS
Transport2TlsDomain = example.org
Transport2TlsClientVerification = Optional
Transport2RecordRouteUri = sip:example.org;transport=TLS
# Configuration for manually maintained TLS certificates:
Transport2TlsPrivateKey = /etc/ssl/private/example.org-key.pem
Transport2TlsCertificate = /etc/ssl/public/example.org.pem
# Configuration for certificates obtained using Let's Encrypt / certbot:
#Transport2TlsPrivateKey = /etc/letsencrypt/live/example.org/privkey.pem
#Transport2TlsCertificate = /etc/letsencrypt/live/example.org/fullchain.pem

# Transport3 will be for SIP over WebSocket (WebRTC) connections
# We use port 8443 here but you could use 443 instead
Transport3Interface = 198.51.100.19:8443
Transport3Type = WSS
Transport3TlsDomain = example.org
# This would require the browser to send a certificate, but browsers
# don't currently appear to be able to, so leave it as None:
Transport3TlsClientVerification = None
Transport3RecordRouteUri = sip:example.org;transport=WSS
# Configuration for manually maintained TLS certificates:
Transport3TlsPrivateKey = /etc/ssl/private/example.org-key.pem
Transport3TlsCertificate = /etc/ssl/public/example.org.pem
# Configuration for certificates obtained using Let's Encrypt / certbot:
#Transport3TlsPrivateKey = /etc/letsencrypt/live/example.org/privkey.pem
#Transport3TlsCertificate = /etc/letsencrypt/live/example.org/fullchain.pem
```

```
# Transport4 is the IPv6 version of Transport3
Transport4Interface = 2001:DB8:1000:2000::19:8443
Transport4Type = WSS
Transport4TlsDomain = example.org
Transport4TlsClientVerification = None
Transport4RecordRouteUri = sip:example.org;transport=WSS
# Configuration for manually maintained TLS certificates:
Transport4TlsPrivateKey = /etc/ssl/private/example.org-key.pem
Transport4TlsCertificate = /etc/ssl/public/example.org.pem
# Configuration for certificates obtained using Let's Encrypt / certbot:
#Transport4TlsPrivateKey = /etc/letsencrypt/live/example.org/privkey.pem
#Transport4TlsCertificate = /etc/letsencrypt/live/example.org/fullchain.pem

# Transport5: this could be for TCP connections to an Asterisk server
# in your internal network.  Don't allow port 5060 through the external
# firewall.
Transport5Interface = 198.51.100.19:5060
Transport5Type = TCP
Transport5RecordRouteUri = sip:198.51.100.19:5060;transport=TCP

# Transport6 is the IPv6 version of Transport6
Transport5Interface = 2001:DB8:1000:2000::19:5060
Transport5Type = TCP
Transport5RecordRouteUri = sip:2001:DB8:1000:2000::19:5060;transport=TCP

HttpBindAddress = 198.51.100.19, 2001:DB8:1000:2000::19
HttpAdminUserFile = /etc/repro/users.txt

RecordRouteUri = sip:example.org;transport=tls
ForceRecordRouting = true
EnumSuffixes = e164.arpa, sip5060.net, e164.org
DisableOutbound = false
EnableFlowTokens = true
EnableCertificateAuthenticator = True
```

Table 11.2, “TLS client verification modes” explains the options that can be used for the `ClientVerification` parameters on each transport.

Table 11.2. TLS client verification modes

`(Transport<num>)TLS-ClientVerification`	**Impact**
`None`	The peer is not asked to send a certificate.
`Optional`	The peer is asked to send a certificate. If the peer sends a valid certificate or if no certificate is sent at all, the connection will be established.
`Mandatory`	Every connection, from client/user devices or external callers must have a client certificate. This is more secure, but some devices don't support client certificates.

If you are not sure, start with the setting `Optional`

As of v1.10.0, *repro* provides a simpler syntax for configuring database access. Create the databases and tables as described in the section called “SQL databases”. To configure *repro* to use your tables, see the examples Example 11.4, “Using *PostgreSQL*” and Example 11.5, “Using *MySQL*”.

Example 11.4. Using *PostgreSQL*

```
DefaultDatabase = 101

Database101Type = postgresql
Database101ConnInfo = host=pg1 port=5432 dbname=repro user=repro password=abc
```

Example 11.5. Using *MySQL*

```
DefaultDatabase = 101

Database101Type = mysql
Database101Host = mysql1
Database101Port = 3306
Database101DatabaseName = repro
Database101User = repro
Database101Password = abc
```

If you need help with the configuration settings, please join the mailing list repro-users [http://list.resiprocate.org/mailman/listinfo/repro-users] and the *reSIProcate* team will try to help.

Use the `htdigest` utility from the Apache web server to set the password for the web interface admin user, as demonstrated in Example 11.6, “Using `htdigest` to set `admin` user password”

Example 11.6. Using `htdigest` to set `admin` user password

```
$ sudo htdigest -c /etc/repro/users.txt repro admin
Adding password for admin in realm repro.
New password:
Re-type new password:
$
```

Now restart the `repro` daemon to use the new settings, as demonstrated in Example 11.7, “Restarting the `repro` daemon (`systemd`)”.

Example 11.7. Restarting the `repro` daemon (`systemd`)

```
$ sudo systemctl restart repro
Restarting repro
```

Testing with `s_client`

You can test that the `repro` daemon is listening on the correct ports and that the certificates are valid by using the *OpenSSL* `s_client` utility. This is demonstrated in Example 11.8, “Using `s_client` to test SIP ports (Debian/Ubuntu)” and Example 11.9, “Using `s_client` to test SIP ports (Fedora/RHEL/CentOS)”.

Example 11.8. Using `s_client` to test SIP ports (Debian/Ubuntu)

```
$ openssl s_client -connect sip-proxy.example.org:5061 \
        -CApath /etc/ssl/certs
...
     Verify return code: 0 (ok)
---
```

Example 11.9. Using `s_client` to test SIP ports (Fedora/RHEL/CentOS)

```
$ openssl s_client -connect sip-proxy.example.org:5061 \
        -CAfile /etc/ssl/certs/ca-bundle.crt
...
     Verify return code: 0 (ok)
---
```

The output should finish with `Verify return code: 0 (ok)`. If you don't see that then the server is not running, the firewall is blocking the connection, the port is wrong or there is some problem with the certificate file or the `repro.config` configuration file. You may also find clues in Syslog or the `repro.log` file.

Login to web administration

Go to `http://server1.example.org:5080` and log in as the `admin` user you created.

Click to add a domain. The screenshot in Figure 11.1, "repro web administration: adding a domain" demonstrates how to add the domain *example.org*.

Figure 11.1. repro web administration: adding a domain

Now you **must** restart the `repro` daemon again to use the new domain, as demonstrated in Example 11.7, "Restarting the `repro` daemon (`systemd`)".

User management

It is not necessary to add users manually through the web interface. If users are identified by TLS client certificates or WebSocket cookie authentication, the SIP proxy does not need to have its own list of users at all. If users are stored in an SQL database, it is possible for other processes to `INSERT` new users into the table and *repro* will see them immediately.

Adding a user

The screenshot in Figure 11.2, "repro web administration: adding a user" demonstrates adding a user called *alice* to the domain *example.org*:

Figure 11.2. repro web administration: adding a user

The list of all users can be easily viewed, as demonstrated in Figure 11.3, "repro web administration: listing users".

Figure 11.3. repro web administration: listing users

Adding routes for numeric dialing

Sometimes, when using a SIP phone, it is easier to dial a number than a full SIP address. In the example.org demonstration, the users have the phone numbers **8001** and **8002**. Figure 11.4, "repro web administration: adding a route" demonstrates how to set up the number **8001** for Alice.

Figure 11.4. repro web administration: adding a route

Once the numbers have been added, it is easy to view them all using the `Show Routes' page:

Figure 11.5. repro web administration: listing routes

It is also possible to test the regular expressions to see how they are evaluated. This can be very useful when using replacement strings (where a single rule automatically maps to many different results), an example is given in Figure 11.6, "repro web administration: routing test".

Figure 11.6. repro web administration: routing test

If you need help with the configuration settings, please join the mailing list repro-users [http://list.resiprocate.org/mailman/listinfo/repro-users] and the *reSIProcate* team will try to help.

Kamailio SIP proxy

Package installation

For Debian/Ubuntu, install the package using `apt`, as demonstrated in Example 11.10, "Installing `kamailio` on Debian/Ubuntu". Fedora, RHEL and CentOS users need to manually add the appropriate repository from *rpm.kamailio.org* [http://rpm.kamailio.org/] and then use `yum` to install, as demonstrated in Example 11.11, "Install `kamailio` on Fedora/RHEL/CentOS".

Example 11.10. Installing `kamailio` on Debian/Ubuntu

```
$ sudo apt-get install kamailio
$ sudo addgroup kamailio ssl-cert
```

Example 11.11. Install `kamailio` on Fedora/RHEL/CentOS

```
$ sudo yum install kamailio
$ sudo addgroup kamailio ssl-cert
```

Configuration

The configuration file is `/etc/kamailio/kamailio.cfg`. The configuration file is written in a scripting language specific to the *Kamailio* project. It is recommended that you look for and adapt an example configuration file that meets your needs rather than trying to write one from scratch.

After you complete the configuration, restart the daemon as demonstrated in Example 11.7, "Restarting the `repro` daemon (`systemd`)".

Example 11.12. Restarting the `kamailio` daemon (`systemd`)

```
$ sudo systemctl restart kamailio
Restarting kamailio
```

Chapter 12. XMPP (Jabber) server installation

Choosing an XMPP server

It is recommended that you use the *Prosody* XMPP server. It is widely used and relatively easy to configure. Latest releases of *Prosody* are being made available as packages on the major Linux distributions.

You can also use *ejabberd*, *jabberd2* or another server if you prefer.

Prosody XMPP server

Package installation

Install the package using the appropriate tool, as demonstrated in Example 12.1, “Installing *Prosody* on Debian/Ubuntu” and Example 12.2, “Install *Prosody* on Fedora/RHEL/CentOS”.

Example 12.1. Installing *Prosody* on Debian/Ubuntu

```
$ sudo apt-get install prosody
$ sudo addgroup prosody ssl-cert
```

Example 12.2. Install *Prosody* on Fedora/RHEL/CentOS

```
$ sudo yum install prosody
$ sudo addgroup prosody ssl-cert
```

Configuration

Prosody can use the certificates created in Chapter 9, *TLS certificate creation*. In the Prosody configuration, you can refer to the PEM files directly under `/etc/ssl`. Alternatively, you can copy the certificate files and private key PEM files to `/etc/prosody/certs` or create symbolic links from that location to the real PEM files. Whichever approach you choose, make sure that the private key file (or a copy of it) is readable by the user that Prosody runs as but be careful to ensure they are not world-readable.

Using the example provided, create configuration files for each domain you want to host under `/etc/prosody/conf.d`. Example 12.3, “Domain configuration file (Let's Encrypt / `certbot`)” demonstrates the minimum configuration for a domain using Let's Encrypt certificates and Example 12.4, “Domain configuration file (manual)” demonstrates the equivalent configuration for a domain using manually maintained certificates.

Example 12.3. Domain configuration file (Let's Encrypt / `certbot`)

```
-- Section for VirtualHost example.com

VirtualHost "example.org"
 ssl = {
  key = "/etc/letsencrypt/live/example.org/privkey.pem";
  certificate = "/etc/letsencrypt/live/example.org/fullchain.pem";
 }
```

Example 12.4. Domain configuration file (manual)

```
-- Section for VirtualHost example.com

VirtualHost "example.org"
        ssl = {
                key = "/etc/ssl/private/example.org-key.pem";
                certificate = "/etc/ssl/public/example.org.pem";
        }
```

Edit the file `/etc/prosody/prosody.cfg.lua`. This is where you can do things like enabling the LDAP authentication module or enabling SQL storage for user data.

Once configuration is complete, restart the daemon as demonstrated in Example 12.5, “Restarting the `prosody` daemon (`systemd`)”.

Example 12.5. Restarting the `prosody` daemon (`systemd`)

```
$ sudo systemctl restart prosody
Restarting prosody
```

User management

It is not necessary to add users manually. If users are authenticated by LDAP, for example, Prosody will dynamically create the XMPP account the first time the user logs in. If such a system is not in use, users can be added manually using the command line utility `prosodyctl` or using a web interface.

Example 12.6, “Using `prosodyctl` to add a user” demonstrates how to add a user at the command line.

Example 12.6. Using `prosodyctl` to add a user

```
$ sudo prosodyctl adduser alice@example.org
```

To use LDAP authentication, make sure that `mod_auth_ldap.lua` is in the Prosody `lib` directory and add the LDAP settings to `prosody.cfg.lua` as demonstrated in Example 12.7, “`prosody.cfg.lua` settings for `mod_auth_ldap`”.

Example 12.7. `prosody.cfg.lua` settings for `mod_auth_ldap`

```
authentication = "ldap"
ldap_server = "ldap-server.example.org"
ldap_rootdn = ""
ldap_password = ""
ldap_filter = "(mail=$user@$host)"
ldap_scope = "subtree"
ldap_tls = true;
ldap_base = "dc=example,dc=org"
ldap_mode = "bind"
```

Further reading

The Prosody web site gives more detailed documentation about setting up the user accounts and other steps [https://prosody.im/doc/install].

ejabberd XMPP server

Package installation

Install the package using the appropriate tool, as demonstrated in Example 12.8, “Installing *ejabberd* on Debian/Ubuntu” and Example 12.9, “Install *ejabberd* on Fedora/RHEL/CentOS”.

Example 12.8. Installing *ejabberd* on Debian/Ubuntu

```
$ sudo apt-get install ejabberd
$ sudo addgroup ejabberd ssl-cert
```

Example 12.9. Install *ejabberd* on Fedora/RHEL/CentOS

```
$ sudo yum install ejabberd
$ sudo addgroup ejabberd ssl-cert
```

Configuration

Modify the file `/etc/ejabberd/ejabberd.cfg`, add the server IP address and the path to the certificate as demonstrated in Example 12.10, “`ejabberd` interface example”.

Example 12.10. `ejabberd` interface example

```
{listen,
 [
  {5222, ejabberd_c2s, [
                        {access, c2s},
                        {ip, {195, 8, 117, 19}},
                        {shaper, c2s_shaper},
                        {max_stanza_size, 65536},
                       %% {certfile, "/etc/ssl/private/example.org-key_combine
                        starttls_required
                       ]},
```

Set up the users: go to **http://server1.example.org:5280**, log in and set up the user accounts.

jabberd2 XMPP server

Package installation

Install the package using the appropriate tool, as demonstrated in Example 12.11, “Installing *jabberd2* on Debian/Ubuntu” and Example 12.12, “Install *jabberd2* on Fedora/RHEL/CentOS”.

Example 12.11. Installing *jabberd2* on Debian/Ubuntu

```
$ sudo apt-get install jabberd2
$ sudo addgroup jabber ssl-cert
```

Example 12.12. Install *jabberd2* on Fedora/RHEL/CentOS

```
$ sudo yum install jabberd
$ sudo addgroup jabberd ssl-cert
```

Configuration

jabberd2's configuration files are in `/etc/jabberd2/` (Debian/Ubuntu) or `/etc/jabberd/` (Fedora/RHEL/CentOS). It can use the certificates created in Chapter 9, *TLS certificate creation*. jabberd2 is modular and each of the components needs to be configured.

The most basic jabberd configuration requires setting the hostname of the server in c2s.xml as demonstrated in Example 12.13, “`jabberd2` c2s.xml example” and the JID domain in sm.xml as demonstrated in Example 12.14, “`jabberd2` sm.xml example” The default storage module in sm.xml and authreg module in c2s.xml are sqlite which works well for most installations.

Example 12.13. `jabberd2` c2s.xml example

```
<id password-change='mu'
   require-starttls='mu'
   cachain='/etc/ssl/public/example.org.pem'
   pemfile='/etc/ssl/private/example.org-key.pem'>
xmpp-gw.example.org
</id>
```

Example 12.14. `jabberd2` sm.xml example

```
<id>example.org</id>
```

Further reading

The jabberd2 web site [http://jabberd2.org] gives more detailed documentation about setting up the server.

Chapter 13. WebRTC

WebRTC, also known as *RTCWeb*, puts two-way media streaming capabilities into the web browser and provides an API to manage them (starting and stopping calls) from the JavaScript embedded in any web page.

The technology has been pioneered in the two major browsers, *Mozilla Firefox* and *Google Chrome*. Other browsers have been following their lead.

There was some instability in the early years of WebRTC but since mid-2014 the technology has stabilised significantly.

There have been some pseudo-WebRTC solutions as well, specifically, browser plugins that offer behavior similar to WebRTC with an emphasis on a specific provider. These solutions are not true WebRTC and they are largely becoming irrelevant now that most users have upgraded to browsers with genuine WebRTC support built in.

WebRTC provides a mechanism for peer-to-peer media streaming (audio or video) but it does not specify the use of any particular signalling system, the mechanism responsible for locating other users and routing calls to them.

Technical overview

Media streaming capabilities

The media streaming capabilities of WebRTC are similar to those used for traditional VoIP and RTC but they differ slightly.

WebRTC mandates support for two audio codecs, *Opus* and *G.711*. In practice, browsers are unlikely to implement proprietary codecs such as *G.729* that are implemented in many desk phones. *Opus* is superior to many of the legacy codecs but it does mean there is a possibility of transcoding with a slight degradation in quality when there is interoperability with legacy technology.

Since November 2014, two video codecs, *VP8* and *H.264* are mandatory for the browser. *VP8* is a royalty-free codec that is likely to be widely adopted by newer technologies. *H.264* exists in many legacy video/webcam/teleconferencing solutions and its presence in the browser enables end-to-end video without transcoding, reducing complexity and CPU requirements and avoiding degradation of the picture.

Despite the support for legacy codecs, many other features of the browser's media stack differ and do not offer direct connectivity to most legacy technology.

A major feature of WebRTC is the use of Interactive Connectivity Establishment (ICE) for effective NAT discovery and traversal. Many legacy technologies, including a lot of softphones and desk phones, do not support ICE or have support for its predecessor, STUN.

The next major feature of WebRTC is encryption. Many legacy phones support basic SDES encryption, with key exchange in the Session Description Protocol `crypto` attributes. WebRTC insists on the use of *DTLS-SRTP* which offers more security but with a complete loss of backwards compatibility.

The final point is the RTP packet itself. Many traditional devices and softphones support *RTP/AVP*. WebRTC requires *RTP/AVPF*.

The combined effect of these small differences is that WebRTC media streams have a higher quality than traditional RTC but they can't interoperate directly with the majority of desk phones and softphones already deployed. Given the extremely wide deployment of web browsers (hundreds of millions of users have already upgraded to browser versions that include WebRTC support) it is envisaged that vendors of related technologies will aim to interoperate with the browser in future versions of their products.

In the meantime, it is necessary for media streams to be passed through some intermediate network component that can transform the streams between the standards. This component is referred to by different names, including *Session Border Controller (SBC)*, *Back-to-Back User Agent (B2BUA)* and *Media Breaker*. In practice, it is relatively easy to configure an *Asterisk* or *FreeSWITCH* server to perform this role.

Signalling protocols

As already mentioned, WebRTC does not provide a signalling protocol for the purpose of locating other users and establishing the media streams to start a call.

JavaScript does provide the *WebSockets* API, a mechanism for asychronously passing messages back and forth between JavaScript and a WebSocket server. Many WebRTC implementors have chosen to use *WebSockets* as a transport for their signalling protocols.

SIP and XMPP, the most common signalling protocols from traditional RTC, have both between adapted to support WebRTC. RFC 7118 specifies the SIP over WebSocket transport and many leading SIP implementations have implemented it. XMPP supports both a HTTP binding and more recently a WebSocket binding specified in RFC 7395. Using one of these protocols is highly recommended for the vast majority of WebRTC projects.

User privacy and security

While media streaming allows for more powerful applications to be deployed through the web, it also creates a greater risk to user security and privacy.

When the JavaScript on a site attempts to activate the webcam or microphone, the browser shows a prompt asking the user to authorize the streaming session. The prompt usually allows the user to choose which devices will be used if they have more than one webcam or microphone/audio source.

Authentication

Authentication between a browser and a server can take place in various ways. If a WebSocket connection is used for signalling and if a user has a client certificate in their browser then it should theoretically be possible to use the certificate to authenticate the WebSocket connection. In practice, this is supported by the *repro* SIP proxy but it is not yet supported by the browsers.

Another possibility is the use of cookies or WebSocket URL parameters. Browser security mechanisms (to protect users from cross-site-scripting) only allow cookies to be used if the web server and WebSocket server have the same domain name. If the servers don't use the same domain, URL parameters must be used instead of the cookie. The web server serving the HTML and JavaScript can send an authentication token to the browser as a cookie and the browser can then present this to the WebSocket signalling server. If using the URL parameter method, the script running on the server-side usually constructs a WebSocket URL and embeds it in the HTML sent to the browser. When the JavaScript activates the WebSocket connection, the request-URL, including the parameters, are sent in the WebSocket upgrade request. Both of these mechanisms are supported in the *repro* SIP proxy.

Standard SIP DIGEST authentication can also be used over the WebSocket transport. One benefit of DIGEST authentication is that challenges can be sent from SIP proxy servers or other network components behind the WebSocket server.

Cookie and URL parameter authentication in repro

To use either of these mechanisms with the repro SIP proxy, simply make sure that there is a value for the `WSCookieAuthSharedSecret` parameter in `repro.config`. If desired, the actual cookie or URL parameter names can be customized, otherwise they use default values.

Example 13.1. `repro.config` settings for cookie and URL parameter authentication

```
WSCookieAuthSharedSecret = some-random-string

# Names of the cookies to use for the cookie authentication protocol
# These are the default values:
#WSCookieNameInfo = WSSessionInfo
#WSCookieNameExtra = WSSessionExtra
#WSCookieNameMAC = WSSessionMAC

# Name of the extension header that must match the content of
# the authenticated WSSessionExtra cookie
#WSCookieExtraHeaderName = X-WS-Session-Extra
```

To send the authentication values as URL parameters, the WebSocket URL passed to JSCommunicator or JsSIP may resemble `wss://ws.example.org:8443/WSSessionInfo=1%3A1429975989%3A142997 6889%3A%2A%40example.org%3A%2A%40%2A;WSSessionExtra=;WSSessionMAC=7bf9ed44bfe7e10153762639419c52fd712de58e`.

Notice that the parameters are sent with the semicolon as a separator, do not send them as query parameters with the ampersand (&) separator. The value of each parameter has to be URL encoded (for example, using the `urlencode()` function in PHP). This is demonstrated in the DruCall source code.

Further details and examples are present in the page on the reSIProcate project wiki [http://www.resiprocate.org/SIP_Over_WebSocket_Cookies].

Practical WebRTC deployment

Although the WebRTC API does not provide a signalling protocol, as described in the section called “Signalling protocols”, this does not mean that deployers need to think about developing something themself. In practice, there are several JavaScript libraries which combine a complete signalling implementation and WebRTC API interaction, giving the web developer a very simple API to start and stop calls.

WebRTC clients and firewalls

Any type of RTC project faces two major problems engaging users: ensuring that users have suitable software and ensuring that there is a suitable network connection. WebRTC comprehensively addresses the first problem, ensuring that users have suitable software, by deploying the software as part of the web browser. This solution is so effective because of the high percentage of users who have web browsers and the vast majority of them are receiving automatic upgrades to the latest version of the browser. The latter problem, ensuring a suitable network connection, is more complicated.

The second problem, network connectivity, is explained in the section called “Getting through firewalls”. The comments in that section are especially relevant to WebRTC. WebRTC clients are particularly well suited to work through these problems because of their native support for ICE, TURN, TLS and HTTP proxy servers.

JsSIP and JSCommunicator

For those interested in using SIP for WebRTC signalling, the most compelling solution now involves a combination of JsSIP and *JSCommunicator* [http://jscommunicator.org]. JsSIP provides the low-level support for SIP message parsing. JSCommunicator provides a high-level API and even a fragment of HTML that can be embedded into an existing page to get up and running quickly.

JSCommunicator works with a *repro* SIP proxy server configured using the settings in Example 11.3, “Sample values for `repro.config`”.

To deploy JSCommunicator, take a copy of the HTML, CSS and JavaScript from an existing web site or from the Github repository. If using the code from Github, it is necessary to download each dependency and then run the script to combine and minify the JavaScript code (see the `README` file). Make any necessary changes to the `config.js` file and embed the `jscommunicator.inc` HTML fragment into an existing page template.

Content Management Systems and other frameworks

Search the plugin or module catalog for any major Content Management System (CMS) and you will find many plugins claiming to offer WebRTC. A large number of these are either promoting browser plugins or promoting proprietary services that do not use open standards and lock you into using a specific back-end. Look for those that offer full source code and support for one of the documented and widely supported signalling protocols, SIP or XMPP, so you will have a free choice to run any of the servers described in this guide or use of any arbitrary provider.

For *Drupal* web sites, adding WebRTC is as simple as adding the *DruCall* [http://drucall.org] module and pointing it to a SIP proxy such as *repro*.

The *DruCall* module is based on JSCommunicator so it also provides a very good example of how to adapt JSCommunicator to other CMS platforms.

Figure 13.1. DruCall/JSCommunicator/JsSIP software stack

Figure 13.1, “DruCall/JSCommunicator/JsSIP software stack” demonstrates the relationship between the JavaScript libraries used in DruCall and the underlying web browser APIs.

Troubleshooting

See the section called “WebRTC and WebSockets”.

Chapter 14. Client devices and softphones

Softphones

There are a variety of softphones available for Linux, Windows and Mac OS.

Many users will get high quality audio by connecting a headset to the 3.5mm audio socket on their computer. On many modern laptops, the 3.5mm connector is a TRRS [https://en.wikipedia.org/wiki/Phone_connector_(audio)#TRRS_standards] socket that integrates the microphone and stereo headphones into a single connection, as used on many popular mobile phones. However, using headsets with a USB connector yields better results in some cases.

The Jitsi [http://www.jitsi.org] softphone is developed in Java and runs on Linux, Windows and Mac OS. It supports both SIP and XMPP.

There is a detailed guide to setup up Jitsi with client certificates [http://www.resiprocate.org/ReproMutualTLSAuthenticationJitsi] in the *reSIProcate* wiki.

The *GNOME* desktop on many common Linux distributions includes the *Empathy* softphone, supporting IM, voice and video over SIP or XMPP.

Various other Linux softphones are available, including *LinPhone* and *Ring* (formerly *SFLphone*).

There are a vast array of Linux messaging applications for XMPP, not necessarily having support for voice or video. Popular choices include *Pidgin* and *Psi*.

One of the most popular proprietary softphones is *Bria* from *Counterpath*.

IP desk phones

A key point to note about desktop phones is that they often have a limited range of codecs and they often support proprietary codecs such as G.729 rather than Internet-optimized codecs such as Opus and iLBC. Given that a significant number of calls will start in a web browser using WebRTC and the Opus codec, codec compatibility is an important consideration.

One of the most recognisable IP phones is the Cisco 7940 and its successors. The Cisco phones are well manufactured with good audio quality (including speakerphone support). However, there are multiple firmware options available and this can be expensive to purchase and complicated to administer. If purchasing used Cisco phones on eBay, it is vital to ensure that you are obtaining proper firmware with the phones or that you have some other legal means of obtaining all firmware. The phones load the firmware and configuration using DHCP and TFTP.

The Polycom Soundpoint IP [http://www.polycom.com] phones are very similar in quality to the Cisco phones but without the licensing complications. They typically support SIP out of the box. The Polycom phones support configuration over HTTPS. From the era of Soundpoint IP 320 and later models, there is a client certificate in each phone. This certificate can be used to authenticate when downloading the configuration, ensuring that SIP passwords can't be compromised. The certificate can also be used to authenticate SIP over TLS connections, this is supported by the *repro* SIP proxy using the settings `EnableCertificateAuthenticator` and `CommonNameMappings`.

Another popular choice is the SNOM [http://www.snom.com] device.

There are various factors to think about when choosing a phone, such as VLAN support, built-in Ethernet hub, power-over-ethernet support, codecs, configuration support, support for NAT traversal (using ICE and TURN) and TLS support.

Smartphone apps

On the Android platform, there are several popular free, open source SIP applications including Lumicall [http://www.lumicall.org] and *CSipSimple* and the popular XMPP client *Conversations*.

All the leading free/open source Android apps can be downloaded from the F-Droid [https://f-droid.org] app store or the Google Play app store.

Users of the Apple iPhone have reported success with the iOS version of Linphone [https://www.linphone.org/] (iTunes download link [https://itunes.apple.com/us/app/linphone/id360065638?mt=8]). A particular challenge for iPhone users is that apps can't keep background connections to arbitrary SIP servers open for receiving incoming calls. This is a restriction that Apple has imposed for all apps. iPhone users may also want to consider a proprietary app such as Bria for iPhone [http://www.counterpath.com/bria-iphone-edition.html].

Click-to-dial

Convenience is a significant factor in the success of any technology. Click-to-dial brings significant convenience to users. Many users will dial a contact from their mobile phone, despite the higher cost of the call, simply because of the convenience of accessing the address book through a touch screen.

Click-to-dial from a computer provides similar convenience.

This section considers various ways to enable click-to-dial.

The Firefox Telify plugin

People will frequently encounter phone numbers in their web browser. They may be browsing an arbitrary web site or they may be accessing a business application that doesn't have any native click to dial support.

Web application developers can markup phone numbers as hyperlinks using the `tel:` URI prefix. This makes the phone number clickable just like a link to another web site or an email address. Unfortunately, few web developers have taken advantage of this feature.

The Firefox plugin *Telify* [https://addons.mozilla.org/en-us/firefox/addon/telify/] solves this problem. It scans the page you are looking at and dynamically converts phone numbers into links that can be clicked.

Mozilla Thunderbird and GNOME Evolution address books

Many people use a productivity suite like *Mozilla Thunderbird* or *GNOME Evolution* for email and address book purposes.

For Thunderbird users, the *TBDialOut* [https://addons.mozilla.org/en-us/thunderbird/addon/tbdialout/] plugin makes phone numbers in the address book clickable as `tel` or `sip` URIs. Dialing is then delegated to the URI handler on the user's system.

For Evolution users, using v3.13.2, there is support for clicking phone numbers and SIP addresses that have been added to the address book. Evolution will only make them clickable if it detects that a URI handler is installed.

Using `sipdialer`

A simple way to handle the `tel` and `sip` URIs is to install the `sipdialer` from *reSIProcate*. The `sipdialer` utility can send a SIP `REFER` to various SIP phones that support this mechanism, including Cisco and Polycom. It is available as a package on Debian, Ubuntu and Fedora.

Using Asterisk or FreeSWITCH

There are various scripts available that can send an instruction over HTTP to an Asterisk or FreeSWITCH server to initiate a phone call. One of these scripts can be installed as a URI handler for `tel` and possibly `sip` URIs on each user's computer.

Chapter 15. Adding ENUM to DNS

ENUM is a scheme described in RFC 6116 for mapping E.164 telephone numbers to SIP addresses, XMPP addresses and various other types of resource. ENUM can enable more rapid discovery of services, network resilience and convenience for people who only have a telephone number or a numeric dialpad.

Telephone numbers are everywhere. People have large collections of phone numbers in their mobile telephone address books. Many organizations have large databases of customer phone numbers that have either been submitted by customers or collected from caller-ID. In public ENUM, the owner of a number is able to give callers specific options for contacting them. In private ENUM, organizations can distribute a rich set of routing data in a highly efficient and convenient format, DNS.

ENUM must be considered from two perspectives: publishing your own data for *ENUM* users and making use of the *ENUM* data published by other people and organizations.

For many people, the first impression of ENUM is the public ENUM tree. Given that many countries have not yet formally adopted ENUM, the public ENUM tree still has some big gaps and is not universally useful for every telephone number that exists in the PSTN. Fortunately, the story does not end there. There are several unofficial ENUM trees and there are many ways that ENUM can be used within an organization.

ENUM is worth looking at for various reasons, including the widespread support in many open source products, the relative simplicity of using distributed and fault-tolerant DNS servers to serve such data and the ease of troubleshooing using basic DNS tools.

How ENUM works

ENUM works with numbers written in the *ITU E.164* notation. For example, a London phone number may be dialed `020 7123 4567` from within the UK but its *E.164* representation is `+442071234567`. *E.164* numbers are always designated with a leading plus (`+`) symbol. The digits following the plus symbol are the country prefix (`44` in the example) and then any area code and finally the local number itself. Any other digits, such as the `0` used when dialing within the UK are not present. You may have noticed this notation is used for incoming phone calls and text messages on your mobile phone.

To perform an ENUM query, the first step is to take the number and normalize it into the *ITU E.164* format. Google's *libPhoneNumber*, a free software project, is an ideal tool for normalizing phone numbers individually or en-masse.

Once you have the *E.164* number, remove the leading plus symbol, reverse the order of the digits and insert periods between them. For the example number, it becomes `7.6.5.4.3.2.1.7.0.2.4.4`.

The next step is to append the ENUM suffix. The public ENUM suffix is `e164.arpa`. To look up the example number in public ENUM, it would be written `7.6.5.4.3.2.1.7.0.2.4.4.e164.arpa`. Private and in-house ENUM services simply use an alternative suffix. You may choose to perform multiple ENUM queries concurrently using different suffixes and then choose the best result.

A DNS query is sent using the query string that has been constructed. The query requests `NAPTR` records.

Finally, the ENUM algorithm must inspect the `NAPTR` records that have been returned and try to identify which, if any, are useful. For example, if the query only returns a HTTP URL, a SIP proxy may not be able to use it and will ignore the result and use some other mechanism to route the call.

Example 15.1, "Using `dig` to perform ENUM queries" demonstrates how to execute an ENUM query from the command line for the UK phone number `01865 332244` in the public ENUM suffix `e164.arpa`.

Example 15.1. Using `dig` to perform ENUM queries

```
$ dig -t naptr 4.4.2.2.3.3.5.6.8.1.4.4.e164.arpa

; > DiG 9.8.4-rpz2+rl005.12-P1 > -t naptr 4.4.2.2.3.3.5.6.8.1.4.4.e164.arpa
;; global options: +cmd
;; Got answer:
;; ->>HEADER ;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 3, ADDITIONAL:

;; QUESTION SECTION:
;4.4.2.2.3.3.5.6.8.1.4.4.e164.arpa. IN NAPTR

;; ANSWER SECTION:
4.4.2.2.3.3.5.6.8.1.4.4.e164.arpa. 86400 IN NAPTR 100 20 "u" "E2U+pstn:tel" "!^
4.4.2.2.3.3.5.6.8.1.4.4.e164.arpa. 86400 IN NAPTR 100 10 "u" "E2U+sip" "!^\\+44
```

Consuming ENUM data

ENUM data can be consumed at various points.

The device or softphone where the user starts a call may have the ability to study the number the user has dialed and search for relevant data in ENUM. The data obtained from ENUM can be presented to the user through the user interface. The Android apps *Lumicall* and *ENUMdroid* both operate in this manner.

SIP proxy servers and Session Border Controllers (SBCs) may also perform ENUM queries when deciding how to route a call. In these cases, the device does not have a user interface to allow the user to choose from multiple results. Typically, the routing system will only look for a SIP address in the ENUM results and will automatically send the call there rather than sending it to a PSTN gateway.

Publishing ENUM data

Setting up *ENUM* as part of your RTC deployment, in the public ENUM tree, an internal ENUM tree or both, are optional steps that provides more flexibility and resilience for routing calls.

Public ENUM

In various countries, it is possible to have your phone numbers registered in the public ENUM tree for `e164.arpa`. Wikipedia maintains a detailed list of national ENUM registries [http://en.wikipedia.org/wiki/Telephone_number_mapping#External_links].

Private and third-party ENUM suffixes

Even if your country does not have a national ENUM scheme yet, many large organizations are operating an internal ENUM service. If you have a domain name server, you can create `NAPTR` records in the zone files just as easily as creating an `A` or `CNAME` record. It is relatively easy to construct a script using Python or Java to read from a company telephone directory and write a zone file for the name server. The file can be regenerated periodically by a `cron` job.

If your phone numbers are available in a local LDAP server, the dlz-ldap-enum [http://www.open-telecoms.org/dlz-ldap-enum] module can be used in a *Bind9* DNS name server to support real-time queries.

Dynamic ENUM from LDAP with `dlz-ldap-enum`

Installation

Install the package using the appropriate tool, as demonstrated in Example 15.2, "Installing `dlz-ldap-enum` on Debian/Ubuntu" and Example 15.3, "Install `dlz-ldap-enum` on Fedora/RHEL/CentOS".

Example 15.2. Installing `dlz-ldap-enum` on Debian/Ubuntu

```
$ sudo apt-get install dlz-ldap-enum
```

Example 15.3. Install `dlz-ldap-enum` on Fedora/RHEL/CentOS

```
$ sudo yum install dlz-ldap-enum
```

Configuration

The configuration file is `dlz_ldap_enum.conf`. On a Debian/Ubuntu system, it can be found in `/etc/bind` while Fedora/RHEL/CentOS users will find it in `/etc/named`.

The first step is to customize the file to specify the exact location of the plugin, the ENUM DNS suffix and your LDAP server connection parameters, as demonstrated in Example 15.4, "Sample `dlz_ldap_enum.conf`".

Example 15.4. Sample `dlz_ldap_enum.conf`

```
dlz "example" {
    database "dlopen /usr/lib/dlz-ldap-enum/dlz_ldap_enum.so 2
        v3 simple {cn=admin,dc=example,dc=org} {secret} {127.0.0.1}
        e164-addr.example.org
        {localhost. root.example.org. 2 604800 86400 2419200 604800}
        localhost
        60
        ldap:///ou=$zone$,dc=example,dc=org???objectclass=top
        ldap:///dc=example,dc=org?mail?sub?telephoneNumber=$record$";
};
```

Now add a reference to this file into the main `named.conf`, which may be under `/etc/bind` or just `/etc` depending upon your system. This is demonstrated in Example 15.5, "Additions to `named.conf` for Debian/Ubuntu" and Example 15.6, "Additions to `named.conf` for Fedora/RHEL/CentOS".

Example 15.5. Additions to `named.conf` for Debian/Ubuntu

```
include "/etc/bind/dlz_ldap_enum.conf";
```

Example 15.6. Additions to `named.conf` for Fedora/RHEL/CentOS

```
include "/etc/named/dlz_ldap_enum.conf";
```

Chapter 16. Troubleshooting

Common problems and solutions

Google Talk/Hangouts users not receiving XMPP chat messages

If you have been using XMPP for a long time, you may well have some friends in your buddy list who are using `gmail.com` addresses. In the beginning, Google largely supported XMPP (with some limitations to TLS support) and it was possible to add `gmail.com` users to the buddy list and communicate with them using IM, voice and video.

As of 2014, Google's XMPP support has become somewhat broken. The messages you send to `gmail.com` contacts using XMPP are never delivered and Google never gives any feedback to indicate there was a problem. You continue to see the contact's status changes in your buddy list, giving no clue that the Google service is broken.

Until the Google issues are resolved, suggest to your contacts that they revert their Google account to use Google Talk instead of Hangouts. Google provides a link for this: `https://plus.google.com/downgrade/`. The XMPP Foundation maintains a list of hosting companies who provide fully functional XMPP support [http://wiki.xmpp.org/web/Jabber_Hosting_Possibilities].

Audio and video quality issues

If possible, try and start with audio alone and see if the issue persists. If the problem exists just using audio, then continue troubleshooting with audio only.

The RTCP protocol, closely related to RTP, provides real-time feedback between the endpoints in an RTP session. Many phones and softphones can provide visual feedback based on RTCP statistics. Look for these statistics and try to identify the rate of packet loss and the latency.

Identify which codecs are in use. If possible, change to a lower bitrate codec and observe if this makes any difference. Some codecs, such as *Opus*, have a variable bitrate and should adapt to network conditions.

If using a softphone, check the computer's metrics, in particular, check that the CPU is not overloaded by other running applications.

Identify the IP address where the RTP packets are being sent, you may be able to discover this from the RTP statistics in the phone or you may need to use a packet sniffer as described in the section called “Packet sniffers”.

Perform a `traceroute` to the remote IP address (see the section called “Operating system utilities”). This will frequently reveal the point where latency is occuring.

If using wifi and `traceroute` reveals latency or packet loss on the first hop, then it is a good idea to try using a cable connection to try and determine if the wifi is troublesome. Wifi can experience congestion if other wifi routers in your building use the same frequency. This can be dealt with by using a utility or smartphone app that analyzes the wifi frequencies in your building to help you find a quiet channel and then manually changing the router settings to use that channel.

If there is latency or packet loss on any hop that includes a cable connection, then you may want to consider replacing the ethernet cable. Sometimes a faulty cable will appear to work sufficiently for applications like email but it will have a horrible impact on real-time audio/video streams. If you have access, log in to the devices on each end of the cable and check the interface statistics, looking for any TX or RX errors. If any interface settings are in automatic mode, set them manually (e.g. set full duplex and set the speed to gigabit).

Another possible reason for latency or packet loss at an intermediate hop may be network congestion or high CPU load on one of the devices in the path. Check the charts for each device, they should be monitored as described in the section called “Monitoring tools”.

Techniques

Monitoring tools

Monitoring can detect problems before end users notice them, help plan for growth and make troubleshooting easier when an unexpected problem does occur.

Use systems like *Ganglia* [http://ganglia.info] and *Cacti* to gather statistics from all servers and network devices. Building up graphs of these statistics allows you to see normal usage patterns and spot any spikes more quickly. These graphs also allow you to visualize what conditions were like during an incident that occurred temporarily.

Ensure log messages are being aggregated by *Syslog*. Using a tool like *LogAnalyzer* [http://loganalyzer.adiscon.com/] to visualize the log messages in colour makes it much easier to spot unusual activity.

Use an alerting tool such as *Nagios* or *Icinga* to monitor the graphs (see *Ganglia-Nagios-Bridge* [https://github.com/ganglia/ganglia-nagios-bridge]) and logs (see *Syslog-Nagios-Bridge* [https://github.com/dpocock/syslog-nagios-bridge])and also monitor the availability of services such as the SIP and TURN processes and the expiry dates of TLS certificates.

Appendix C, *Configuring Nagios to monitor SIP, XMPP and TURN* contains specific instructions to configure Nagios/Icinga to monitor SIP, XMPP and TURN servers.

Check the logs

Check `/var/log/syslog` and `/var/log/daemon.log`.

Many of the RTC processes can also create their own logfiles in other locations. Refer to the configuration files to see which type of logging each process uses.

The `repro` daemon web interface has a control on the *Settings* page where the log level can be changed at runtime without restarting the daemon.

Check the web interface

For those processes that have a web interface, this can be a useful way to see runtime activity at a glance.

The `repro` daemon web interface has a *Registrations* page where all known SIP registrations can be seen.

Operating system utilities

Example 5.2, “Inspecting DNS entries with `dig`” demonstrates how to use the `dig` utility to verify that the DNS entries exist.

Use the `netstat` or `lsof` commands to verify that each process is listening on the correct IP addresses and ports.

If a process is failing to start and the reason is not clear in the log file or console output, use the `strace` utility to determine whether any *syscall* is failing.

the section called “Testing with `s_client`” demonstrates how to use the *OpenSSL* `s_client` utility to make a test connection to the SIP proxy on the TLS and WebSocket over TLS ports.

Packet sniffers

Use `tcpdump` to capture the packets to a file on the server. Copy the capture file to a workstation and inspect it with *Wireshark*.

For SIP over UDP or TCP, without encryption, the `ngrep` utility can be useful for identifying packets containing a particular string.

Debugging mode

Run the process in debug mode, in the foreground on a terminal, to see what it is doing. Running the `repro` daemon this way will allow you to see the SIP messages in the console.

WebRTC and WebSockets

Both major browsers have a built-in troubleshooting system for WebRTC. In the *Mozilla Firefox* browser, enter the URL `about:webrtc`. In the *Google Chrome* browser, the URL is `chrome://webrtc-internals`.

Chapter 17. PBX Setup

A soft PBX (such as Asterisk or FreeSWITCH) provides features such as voicemail, menu systems and call queues. Many of the typical features in a soft PBX have a particular focus on voice communications, rather than other types of RTC such as IM or video.

It is perfectly feasible to build an RTC environment without these features. It is recommended that a SIP proxy is used to handle all connectivity with users and any external parties, the reasons for this are explained in the section called “All SIP connectivity through a SIP proxy”. This also means that it is better to install and test the SIP proxy before making decisions about the selection and design of the soft PBX.

The soft PBX can be configured to make a SIP registration in the SIP proxy or routing entries can be created in both the SIP proxy and soft PBX to send calls back and forth between them as required.

The configuration and operation of soft PBXes tends to replicate many concepts from the world of traditional telephony. The fact that many of the features of these products are voice-orientated is an example of this trend. Many guides on soft PBX configuration tend to focus on the use of dial plans and phone numbers as the dominant currency in the world of legacy communications, while modern unified communications strategies involve `user@domain` addressing similar to other Internet-based services.

The all-in-one myth

It is tempting to look at a soft PBX as a one-stop-shop for VoIP, with no other product required. In fact, this is a very limited strategy in terms of features and resilience.

Many people have a requirement to use existing ISDN lines with their RTC architecture. With Asterisk and FreeSWITCH both boasting ISDN support, it is tempting to do everything with that single instance of a soft PBX installed on a server with ISDN hardware.

Modularity is a hallmark of good design in any IT project. Modularity gives you the flexibility to upgrade or modify one component of a system without changing any other component. Modularity also means that some components are more likely to continue providing service even if something fails. In the planning of a soft PBX deployment, modularity involves running one instance of the soft PBX just to control the ISDN hardware and running another instance for services such as voicemail and using a SIP proxy to route calls between these different components.

Choosing between Asterisk and FreeSWITCH

Asterisk is by far one of the most well known open source VoIP server products. It has a plethora of features and supports a diverse range of connectivity options, from IP based solutions like SIP to legacy technologies like ISDN and POTS.

FreeSWITCH is a popular alternative to Asterisk, boasting many of the same features, but with a collective ownership model rather than the corporate model that is intertwined with Asterisk's licensing and contributor agreements.

We consider some of the points where they differ. Not every point is relevant in every deployment.

If you are not sure which way to go and if maintaining the soft PBX will not be the primary focus of your job, we suggest using Asterisk because it has official packages but we do not wish to discourage people from considering FreeSWITCH when they understand the issues involved and feel it is the better choice.

Official packages

Asterisk has officially supported packages in many of the popular GNU/Linux distributions. FreeSWITCH does not.

FreeSWITCH users have to use unofficial packages from the FreeSWITCH web site or compile from source code. When packages are made available through an official distribution, like Debian or EPEL, it means they go through a managed release cycle and are subject to a battery of independent tests to ensure the packages don't interfere with any other packages on the same system.

When something is available in an official package, it also means upgrading the operating system (for example, from Debian 7 to Debian 8) should also upgrade the package in a safe manner.

Contributing patches

Many more experienced users of open source software become intimitely familiar with the software they rely on and sometimes they even develop bug fixes or improvements. It can be tedious to test and re-apply such fixes to each new version of a product and so many people in this situation usually want to contribute their fix to the primary source repository for the project so it will automatically be part of all future releases.

Some projects welcome these contributions unconditionally and without any specific legal agreement, ownership of the contribution remains the intellectual property of the contributor and other members of the community are simply licensed to use it.

Some projects ask the contributor to go a step further, giving the founders of the projects some additional rights to include the contribution in commercial products without releasing the source code of the final product.

Finally, some projects go all the way and don't just ask the contributor for a preferential license, they ask the contributor to grant ownership of all intellectual rights in the contribution to the project's founders or some other entity.

Asterisk is in second category, asking contributors to give Digium, the company behind Asterisk, a license to redistribute the contributions under alternative terms. Some projects use this strategy to collect intellectual property rights in a non-profit community foundation but in this case the rights are being granted to Digium, a company with shareholders. Another point of contention is that the agreement is one-sided: it does not contain any commitment by Digium to release future versions of the product under any open source license.

Some contributors are uncomfortable with these contribution terms and some people are unable to make contributions under these terms without violating regulations set by their own employer or funding they receive from public sources.

Licensing

Asterisk is distributed under the GPL while FreeSWITCH is using the Mozilla Public License.

The main distinction between these licenses applied to those users who intend to build their own product around the soft PBX and sell it. The GPL typically requires people in this situation to either publish the source code of any fixes or enhancements or other components they create as part of their overall solution. Digium has a parallel-licensing strategy, allowing people in this situation to pay a license fee and eliminate the obligations of the GPL.

The Mozilla Public License used by FreeSWITCH doesn't involve these issues.

Community

The contributor agreement and the licensing strategy have an impact on the type of community that is forming around each product, especially the community of developers contributing to the products.

Even for users who don't intend to either contribute source code or resell products built from Asterisk or FreeSWITCH, it is important to consider which community is more sustainable in the long term.

Many people have their own thoughts and feelings about this. It may well be that the existence of both competing products with distinct communities is the most sustainable solution for both.

Scalabiltiy and code quality

One of the reasons that FreeSWITCH was founded is because the creators were not satisfied with the design of Asterisk and did not feel that Asterisk would change. FreeSWITCH was written from the ground up to address some of those perceived problems.

One area where such problems exist is in the case of scalability. FreeSWITCH's design is more favorable to large scale operation.

It should be noted that while the designers of FreeSWITCH have chosen to emphasize different priorities and achieved a more scalable solution, Asterisk has involved in other ways and some people feel the range of features Asterisk offers for developing their customized applications is superior and more relevant to them than scalability.

Using Asterisk with the repro SIP proxy

Setup the SIP proxy as described in the section called “repro SIP proxy”. Add a UDP transport in `repro.config`, it will be used to communicate with Asterisk. It is a good idea to ensure that the firewall prevents external hosts from sending any UDP traffic to either the SIP proxy or the SIP port on Asterisk.

Go to the repro web administration page and click to add a route. Configure the route using the details in Table 17.1, “repro route configuration”. Replace `A.B.C.D` with the IP address of the Asterisk server.

Table 17.1. repro route configuration

Parameter	Value
URI	`^sip:([0-9]*)@sip-proxy\.example\.org;.*transport=tls`
Destination	`sip:$1@A.B.C.D:5060;transport=udp`

Notice that this route looks for the `transport=tls` parameter. This means it will only be applied to calls coming from the users. When a call comes into the SIP proxy from the Asterisk server, it will be coming over the UDP transport and it won't be matched by this route (otherwise it would go into a loop).

Next, in the repro web administration page, click to add an entry to the ACL. Add an entry permitting packets from the IP address and SIP port used by the Asterisk server.

Remember to restart the repro SIP proxy if changes were made to the list of domains or the `repro.config` file.

In the Asterisk server, setup the `sip.conf` file as demonstrated in Example 17.1, “Asterisk `sip.conf`”. In particular, replace `A.B.C.D` with the IP address that the Asterisk server should bind to, this must be a routable IP address on the Asterisk host.

Example 17.1. Asterisk `sip.conf`

```
; should match the realm used by the proxy
realm=sip-proxy.example.org
```

```
domain=sip-proxy.example.org
fromdomain=sip-proxy.example.org
port=5060
bindaddr=A.B.C.D

; follow this pattern to define a user
[8001]
username=8001
secret=whatever
host=dynamic
canreinvite=no
mailbox=8001
nat=yes
```

The Asterisk server will need to be able to send calls to SIP users who are registered with the SIP proxy. The calls can be routed using the `Dial` command as demonstrated in Example 17.2, “Asterisk `extensions.conf`”.

Example 17.2. Asterisk `extensions.conf`

```
exten => _8XXX,n,Dial(SIP/${EXTEN}@sip-proxy.example.org,45)
```

Chapter 18. PSTN connectivity

The Public Switched Telephone Network (PSTN) remains pervasive and convenient for much of the world's population, even though it is also expensive, unsophisticated and insecure.

In the early days of IP telephony, the PSTN was perceived as something of a security blanket. Companies would design their IP telephony system to mirror the traditional PBX. Some would even keep a few analog phone lines connected in case of an "emergency".

This fear-based approach has gradually been replaced by a more pragmatic approach, relegating the PSTN services to being just one part of the big picture. A modern RTC-based solution works seamlessly with or without the PSTN.

The emotional attachment to the PSTN has largely been contained due to two factors: the widespread presence of smartphones as an alternative means of communication that can be used in an emergency and the increased investment in corporate IT networks which need to be always available.

This chapter looks at PSTN connectivity issues.

Methods of PSTN connectivity

Anybody, anywhere in the world, can now rent inbound phone numbers and make outgoing calls using SIP trunking providers. This is often the fastest, cheapest and most flexible means of getting connected to the PSTN. It is often possible to port existing phone numbers from legacy phone companies to SIP operators.

There are a range of hardware options for joining analog and ISDN telephone lines to an IP network. One option is to purchase a dedicated media gateway, such as the those promoted by Cisco Systems and other well known vendors. Another option is to install an ISDN or analog telephony card into a server and run Asterisk or FreeSWITCH, as described in Chapter 17, *PBX Setup*, configured solely to act as a media gateway.

If dedicated ISDN links to the telephone company are important, it is worthwhile considering one further permutation: instead of installing the media gateway at the office location, have the ISDN lines installed to a rack in a data center and use VPN or WAN links from the data center to the office. This can achieve higher resilience and flexibility: if the main office site suffers from any kind of technical or environmental disturbance, calls can be routed from the data center to users at home, on mobile devices or at another branch office or disaster recovery location.

ingress call handling

The concept of an *ingress* module accepting calls from other places such as SIP trunks or ISDN trunks is introduced in the section called “Routing calls within a site”.

The *ingress* stage should, in most cases, normalize both the destination number and the caller ID into the E.164 format. This will assist with both routing and log analysis later on.

The *ingress* stage may also convert the destination numbers to the addresses of internal users such as `user@example.org`. This can also be done at the routing stage.

egress call handling

The concept of *egress* modules preparing calls from the local users to go out to SIP trunks or ISDN trunks is also introduced in the section called “Routing calls within a site”.

The *egress* stage should convert the destination number into the format expected by the telephone company or trunking provider. For example, they may require a `00` prefix on all numbers qualified with a country code.

The *egress* stage is also a good place to set the caller ID. Some trunking providers automatically set the caller ID for you. Some allow you to specify the caller ID in the `From` header, as long as the value you use is an inbound number that you have purchased from the same company. If you try to use any other number, it is likely the will replace it with one of your numbers selected at random. ISDN trunks attached to a soft PBX or media gateway also permit the caller ID number to be set on a per-call basis.

Setting the caller ID to a constant value is relatively easy within an Asterisk extension as demonstrated in Example 18.1, “Asterisk `extensions.conf` for specifying caller ID”.

Example 18.1. Asterisk `extensions.conf` for specifying caller ID

```
exten => _X.,1,Set(CALLERID(name)=00442071358378)
exten => _X.,2,Set(CALLERID(number)=00442071358378)
```

Instead of hard-coding a constant value, if the users have their own direct phone numbers, a CSV file or Asterisk Realtime SQL lookups could be used to map individual SIP usernames to personal caller-ID values.

Emergency calls

Providing access to dial the emergency services has been a controversial issue for all IP-based telephony. Users may see a phone and want to use it to make such a call. Anybody installing phones should contemplate what happens in this situation.

The first thing to note is that not all organizations route emergency calls to the publicly operated emergency call center. Some large organizations and universities route these calls to their internal security office. The suitability of this approach depends on local laws and the training of the security staff.

Many SIP providers now offer an option to handle emergency calls. For this to be effective, the SIP provider usually needs to have accurate records of the addresses where the SIP trunks are used. For example, care needs to be taken to ensure that emergency calls are not routed from mobile VoIP users when they are off-site, as the emergency services may arrive at the wrong location.

Another option is to have a single physical phone line or GSM SIP gateway attached to the IP phone system solely for routing emergency calls. All other calls would be routed to the normal SIP providers.

If it is not technically feasible to route calls to the emergency service number, it may be useful to provide a brief recorded message telling the user to call emergency services from another phone. After playing the message once or twice, the PBX should hang up, so the user knows that the call is not being routed.

In the early days of telephony, not every home or office had a telephone. In Britain, which is known for its distinctive red phone boxes, the police force took responsibility for installing many *blue* phone boxes for use by police and people without a telephone. Some people feel that it is important for the emergency services to be proactive again and ensure that emergency call centers have a presence on the Internet, accepting calls directly from users of SIP and XMPP.

Chapter 19. Frequently Asked Questions

Can I use a virtual server for SIP or XMPP?

For small installations (less than 30 concurrent phone calls) a virtual server running on modern hardware is more than sufficient.

For demanding use-cases, it is recommended that any real-time processes (such as the TURN server or a soft PBX like Asterisk or FreeSWITCH) be on dedicated servers while it may still be possible for the SIP proxy or XMPP server to be on a virtual server.

Do I really need to use TLS encryption and certificates?

Yes. Even if you don't care too much about security or privacy, TLS helps to reduce the risk of nuisance calls from spammers and the risk of impersonation and it also eliminates a range of problems caused by SIP-aware routers that try to modify SIP messages to help them through NAT.

Chapter 20. Community support

Many technologies and companies have a presence on the Internet for support and interaction between between users and developers. This model of community support is particularly relevant for RTC technology. RTC, by definition, requires interoperability between all those using the technology and efficient online collaboration helps to identify and resolve interoperability problems. Many users also have common problems to solve, such as the collaboration between Internet companies to eliminate spam.

In the RTC domain, there are a range of online communities with various objectives. The objectives include supporting specific pieces of software, development of standards, discussion of operational issues between providers and advocacy of open standards and private communication.

Mailing lists

Major announcements

The Free-RTC-Announce mailing list [http://lists.freertc.org/mailman/listinfo/announce] is a low-volume mailing list where you can receive important announcements about Free RTC products, community activities and events. The list is fully moderated and aims not to send more than one announcement per week. Please go ahead and subscribe now [http://lists.freertc.org/mailman/listinfo/announce].

Strategy and advocacy

The FSFE Free RTC mailing list [https://lists.fsfe.org/mailman/listinfo/free-rtc] discusses strategies for promoting and adopting free RTC and free software.

Collaboration between operators and service providers

The XSF XMPP operators mailing list [http://mail.jabber.org/mailman/listinfo/operators] enables discussion of issues between organizations managing XMPP servers of any size.

Server support

The reSIProcate `repro` users mailing list [http://list.resiprocate.org/mailman/listinfo/repro-users] provides community support for people setting up and operating the `repro` SIP proxy server.

The Prosody users mailing list [http://prosody.im/discuss#mailing_lists] provides community support for people setting up and operating the *Prosody* XMPP server.

Softphones

The Lumicall users mailing list [http://lists.lumicall.org] supports users of the *Lumicall* Android app.

The Jitsi users mailing list [https://jitsi.org/Development/MailingLists] provides a resource for those setting up and using the Jitsi products, including the popular Jitsi softphone.

Popular blogs and news sites

planet.sip5060.net [http://planet.sip5060.net] syndicates blogs from leading software developers with a focus on RTC and especially SIP.

Appendix A. Building reSIProcate packages on Debian/Ubuntu

reSIProcate packages are usually available in *stable-backports*. Sometimes, such as during a Debian release freeze, *stable-backports* won't contain the latest upstream version or there is some other reason to build the package from source. This is a relatively straightforward task using the `debuild` utility.

Example A.1. Installing the `debuild` command

```
$ sudo apt-get install devscripts
```

Example A.2. Installing the compiler and dependencies

```
$ sudo apt-get build-dep resiprocate
```

Example A.3. Running the `debuild` command

```
$ wget http://.../resiprocate-1.9.8.tar.gz
$ tar xzf resiprocate-1.9.8.tar.gz
$ cd resiprocate-1.9.8
$ debuild -rfakeroot -i -us -uc -b --no-lintian
```

Example A.4. Running the `debuild` command using code from Git

```
$ git clone https://github.com/resiprocate/resiprocate
$ cd resiprocate
$ git checkout 1.9.8
$ debuild -rfakeroot -i -us -uc -b --no-lintian
```

Appendix B. Building reSIProcate RPMs on RHEL and CentOS

reSIProcate packages are available on many recent Linux distributions. On some platforms, including Red Hat Enterprise Linux and CentOS, it may be necessary to build the RPMs from source. This is a relatively straightforward task.

Example B.1. Installing the `rpmbuild` command

```
$ sudo yum install rpm-build
```

Example B.2. Installing the compiler and dependencies

```
$ sudo yum install gcc-c++ libtool automake autoconf \
    asio-devel boost-devel cajun-jsonapi-devel c-ares-devel \
    cppunit-devel gperf db4-cxx-devel db4-devel openssl-devel \
    mysql-devel pcre-devel perl popt-devel python-devel \
    python-pycxx-devel freeradius-client-devel xerces-c-devel
```

Example B.3. Creating the `rpmbuild` directories

```
$ mkdir ~/rpms
$ cd ~/rpms
$ mkdir BUILD BUILDROOT \
        RPMS/i386 RPMS/noarch RPMS/x86_64 \
        SOURCES SPECS SRPMS
$ cat > ~/.rpmmacros << EOF
%_topdir $HOME/rpms
%__make               /usr/bin/make -j7
EOF
$
```

Example B.4. Running the `rpmbuild` command

```
$ wget http://.../resiprocate-1.9.8.tar.gz
$ rpmbuild -tb resiprocate-1.9.8.tar.gz
```

Appendix C. Configuring Nagios to monitor SIP, XMPP and TURN

Users become very frustrated with a service if it is not working when they need it. Monitoring systems like Nagios help to discover any outage at the first opportunity and inform the right person to fix it.

Nagios plugins

It is necessary to download and install the individual Nagios plugin scripts. Once that is done, the plugins must be declared in the Nagios configuration. They can be used over and over again to create configurations for monitoring different services on different hosts.

As the TURN server uses the STUN protocol, a STUN test script is sufficient to test the TURN server is operational.

Example C.1. Sample `/etc/nagios-plugins/config/stun.cfg`

```
# can be used to check STUN and TURN servers
# uses script from http://karlsbakk.net/asterisk/scripts/check_stun
define command {
        command_name check_stun
        command_line /usr/local/lib/nagios/plugins/check_stun $HOSTADDRESS$
}
```

Example C.2. Sample `/etc/nagios-plugins/config/sip.cfg`

```
# uses script from https://github.com/ibc/nagios-sip-plugin
define command {
        command_name check_sip_tls
        command_line /usr/local/lib/nagios/plugins/check_sip2 -t tls -p $ARG2$
}
```

Example C.3. Sample `/etc/nagios-plugins/config/xmpp.cfg`

```
# uses script from https://exchange.icinga.org/jandd/check_xmppng
# Debian/Ubuntu: apt-get install nagios-check-xmppng
define command {
        command_name check_xmpp
        command_line /usr/lib/nagios/plugins/check_xmppng -H $HOSTADDRESS$ --se
}
```

Nagios service checks

Once the plugins are declared in the Nagios configuration, they can be used to write service check definitions as demonstrated here.

Example C.4. Sample service check for STUN/TURN

```
define service{
        use                             generic-service
        host_name                       turn-server.example.org
        service_description             STUN/TURN
        check_command                   check_stun
}
```

The following test SIP on port 5061 and 443:

Example C.5. Sample service check for SIP over TLS

```
define service{
        use                             generic-service
        host_name                       server1
        service_description             SIPS
        check_command                   check_sip_tls_port!sip-server.example.o
}
```

Example C.6. Sample service check for SIP over TLS (port 443)

```
define service{
        use                             generic-service
        host_name                       server1
        service_description             SIPS 443
        check_command                   check_sip_tls_port!sip-server.example.o
}
```

The following monitors the XMPP service. Notice that the argument used for this check must be the XMPP domain, not the server domain or hostname. In the example, it is *example.org*.

Example C.7. Sample service check for XMPP

```
define service{
        use                             generic-service
        host_name                       server1
        service_description             XMPP
        check_command                   check_xmpp!example.org
}
```

Appendix D. Mitigating VoIP fraud risk

VoIP fraud is not a new problem, it is just an old problem with a new target. Fraud is not a reason to avoid VoIP and RTC technology. Fraud has been a regular problem for companies with traditional ISDN phone systems. Managing the risk requires a balanced approach.

Legal insurance

In the event that your phone account is misused, you may end up in a legal dispute with your phone company. Check that you have a satisfactory legal costs insurance policy. Verify that the terms and conditions include cover for disputes with utility companies.

Trade body membership

If you are operating a business, are you a member of any trade organizations, such as the local chamber of commerce?

These organizations sometimes provide useful advice and sometimes arrange legal insurance on behalf of members.

Set a credit limit

If you leave the cookies out on the table, it won't be long until children start eating them and they will keep eating until they are all gone. Likewise, if your VoIP PBX is hacked, the bad guys are going to use it to relay calls to high cost destinations: *and they are not going to stop until you switch the system off or the phone company cuts the line*.

The number one thing you can do to protect your phone system does not involve any technical changes. It simply involves writing a letter.

Write to your phone company and tell them the amount of daily and monthly expenditure you authorise. Make it clear that this is both a security precaution and that any severe violation may jeopordise your business to the extent that you may not be able to pay bills in future.

> Dear Sir,
>
> My phone number is ___________ and my account number is ____________
>
> I am writing to inform you that the total authorized expenditure for this account is $_______ per day and $________ per month.
>
> Any services supplied in excess of this authorization will be treated as if they were supplied in error and we accept no liability for them.
>
> Furthermore, I am informing you that we explicitly do not require the use of any of the services listed below, that the supply of these services is not authorized and that if any of these services are supplied to us or billed to us without management authorization, it is an error of the phone company and therefore it will not be paid.
>
> - Calls to premium rate numbers
> - Reverse charge calls charged at a rate in excess of $___ per minute
> - Calls to numbers where a share of the call charge is paid out to the recipient of the call (such as the UK 0871 numbers)

- Premium-rate text messaging services
- Data roaming charges
- Data charges for data usage in excess of bundled data allowances

This letter has been sent to you by recorded delivery and takes effect on receipt.

Sincerely,

____

Director/Manager/President

Use a different phone company for inbound numbers

If you do ever end up with an inflated bill that has come about because of illegal use of your VoIP system, and if your phone company has somehow lost the letter you sent specifying your maximum authorized expenditure, they might try and bully you into paying the bill anyway.

Phone companies have large accounts departments that are very experienced at manipulating and bullying customers to pay a bill whether it is correct or not. A recent report by analysts Juniper Research suggested that phone companies lose over $58 billion per year due to their own technical faults in billing technology. The magnitude of this figure emphasises one particular point: phone companies may sometimes underbill you, they may sometimes overbill you, but if customers were watching their bills more closely, phone companies wouldn't be making so many mistakes.

With such unreliable systems, the phone company has very little evidence they can rely on to force you to pay a bill. So they simply cut off customer's numbers.

This is why this point is so vital: use two different phone companies.

All your outgoing calls go through one company (company A).

All your phone numbers and incoming calls come through a different company (company B).

If your VoIP system is hacked or misused in some way, company A might cut off your line - but you will still be receiving incoming calls normally thanks to company B.