Desktop sharing with SIP

Author : Willem Toorop *
Supervisor : Michiel Leenaars †

February 2, 2009

Abstract

This report describes how Desktop and Application sharing sessions can be realised using SIP. Investigated is what possibilities require the least or no adaptation of existing SIP infrastructure. An implementation of the RFB protocol tunneled over a MSRP session using the sipsimpleclient library is presented as a possible solution.

The research question this report tries to answer is: *"How can application and desktop sharing, initiated by SIP, be realised in existing SIP infrastructure with the least possible impact on that infrastructure"*

Many thanks to Adrian Georgescu who came up with the RFB over MSRP idea.

UNIVERSITY OF AMSTERDAM

*willem.toorop@os3.nl
†michiel@nlnet.nl

Contents

1 Introduction

In this section I will describe: what application and desktop sharing is, what SIP is, and why, from a user perspective, it would be nice to be able to set up desktop and application sharing sessions with SIP.

1.1 What is application and desktop sharing?

In this report, when we talk about Application Sharing we mean the sharing of the graphical user-interface of an application amongst multiple users simultaneously in real time. The user-interface is the window through which the application communicates with the end-user.

A shared application is running on a host computer, which can be the computer of one of the end-users. Generally this user will provide access to the application to other users, but different setups are imaginable; The host computer could be a server dispensing new instances of a certain application to end-users on request. A company selling an application could, for example, provide evaluation services of the application via application sharing.

But, application sharing is not only remotely starting an application and getting the user-interface presented locally. This feature, which in the context of the X-Windows system is called Network Transparency, is a subset of application sharing functionality. With application sharing we not only have remote access to an user-interface of a new started application, but also of an already running application.

With Desktop Sharing, in stead of a single application shared amongst multiple users, a whole screen is shared. Since the desktop, and all application windows on it, are run by a controlling application, desktop sharing can also be viewed as application sharing of that controlling application. On a X-Windows system this controlling application is the X server usually started by the `X` command. On a MS-Windows system the application running the desktop is `explorer.exe`.

1.2 What is SIP?

SIP, the *Session Initiation Protocol* [1], is an IETF protocol for setting up sessions between end-users. SIP makes sure the end-users can be found wherever they are on the internet. The end-users can be found by their address-of-record, which usually is the same as their email-address. SIP also helps in negotiating the type of session that the end-users want to set up. These sessions are often multimedia sessions such as a voice or video calls, but SIP itself is not responsible for those sessions. It just finds the end-user and then makes sure the type of sessions is agreed upon.

The type of those sessions are certainly not restricted to voice or video

calls, but can in fact be anything; Although those types do have to be registered with IANA[1] as MIME media-types. See: http://www.iana.org/assignments/media-types/index.html.

The end-users makes use of a so called User Agent (UA), to setup a session with another end-user. The User-Agent can be a computer program, but can also be a hand-held device such as a telephone. End-users can have multiple User Agents which each support different sessions. SIP ensures the session is set up with the User Agent supporting that session type.

With SIP, end-users can also use features usually seen in telephony networks: End-users can forward their address to another address; They can transfer an existing session to another end-user; They can invite other end-users to participate in an existing session (like in a conference call).

SIP, at the moment, has a widespread use in VOIP and Internet Telephony. Many internet providers currently offer telephony services using SIP and there is an increasing number of telephony providers offering cheap rates to the public telephone network (PSTN) via SIP[2].

The main difference between SIP and other standards, is that SIP users a relatively simple infrastructure and most features are in the User Agent. Other signaling standards, such as SS7, use dumb end-points and concentrate all their functionality in their network architecture. This makes it very easy to add new features to SIP. One only has to adapt the User Agent used; All the existing SIP infrastructure doesn't need any adaptation.

SIP finds also popular usage in Instant Messaging (IM). Microsoft's Messenger version 4 and 5 support SIP and many free and opensource software IM clients support SIP; Like SIP Communicator. Often those clients combine Instant Messaging functionality with VOIP and video conferencing capabilities.

1.3 Why application/desktop sharing with SIP

There are many application and desktop sharing products available. Some are installed with the operating system, such as Terminal Services with Microsoft Windows, using the RDP protocol. Others use a publicly available protocol such as the products based on the RFB[2] protocol; Like RealVNC

Often they are incorporated within an Instant Messaging client. Microsoft's Netmeeting supports application sharing, and Apple's iChat supports screen (desktop) sharing, but these are closed source products, only available on the operating system they are build for. They also have the disadvantage of using proprietary protocols, or modified open standards for IM, application/desktop sharing and finding the participating party.

A application/desktop sharing solution using SIP would have the advantages that:

[1] The body responsible for the coordination of internet protocol resources.
[2] See for example http://sip.startpagina.nl/ under "SIP Telcos"

- SIP is an open standard. This stimulates the development of SIP supporting products.
- You have a wide variety of SIP account providers, which are completely independent of the User Agent used. It is also possible for organizations to implement there own SIP infrastructure (which is relatively simple), and even makes them independent from a SIP account provider.
- End-users can be contacted by their SIP address, which can be the same as their email-address and is thus easy to remember.
- It inherits all SIP's features. (Redirection, conference calls, call forwarding)
- The application/desktop sharing session can be combined with other session types using the same SIP connection. (IM, audio, video etc.)

One of he initial goals of this research project was to create an application sharing with SIP implementation. Unfortunately I haven't succeeded in realizing *application* sharing, although I have good references and clues for how this can be realised. I did manage to get a working desktop sharing implementation with SIP. Therefor the title of this report is "Desktop sharing with SIP." Still, the infrastructural needs for both are explored in this report, and in fact are more or less the same.

The research question this report tries to answer is: *"How can application and desktop sharing, initiated by SIP, be realised in existing SIP infrastructure with the least possible impact on that infrastructure"*

2 How does SIP work?

In this section a very basic overview is given of the workings of SIP.

2.1 Overview

There are two kind of SIP addresses. One is the regular SIP address, an end-user can be contacted with by another end-user. This regular SIP address is called a address-of-record, and is formated as a URL in the form `sip:user@domain`. The other one, is not to be used by end-users, but for the internal workings of the SIP network elements; It is a so called SIP contact-address and denotes a host, port, protocol, and option id servicing the SIP protocol. It is also formated as a URL and is in the form:
`sip:id@host:port;transport=protocol;parameters`.
Extra parameters can follow a contact-address.

SIP infrastructure consists of four different types of network elements: User Agents, Registrars, Proxies and Redirect servers. The User Agent is

the client program used by the end-user. The other elements are more a role in the SIP infrastructure, then services running on separate hosts. One host could be, and often is, a Registrar, a Proxy and a Redirect server in one. The network elements are described in the following subsections.

2.2 Registration

The purpose of the registration is to bind a address-of-record to one or more contact-addresses. By looking up the contact-address for an address-of-record, SIP network elements know where to contact the end-user's UA. This is how the location-independency of SIP is realised.

When an end-user first starts its User Agent, it first registers with a SIP Registrar. After registration the Registrar (and other SIP network elements using the same Location Service), knows what contact-address should be contacted for a specific address-of-record.

The IP-address and port number, on which to contact the registrar, is either configured in the UA, discovered by DNS lookups[3] or via a request to the "all SIP servers" multicast address: `sip.mcast.net` (`224.0.1.75`).

2.2.1 Finding the registrar

When using DNS, first a NAPTR record[4] query is done on the domain part of a end-user's sip address. NAPTR records are the replacement for service specific record, such as MX.

The end-user address-of-record might for example be `sip:michiel@isoc.nl`. We first then have to query a NAPTR record for the `isoc.nl` domain. The returned value was at the time of writing:
`10 0 "S" "SIP+d2u" "" _sip._udp.isoc.nl.`.

The fields in this returned value are:

Order	Preference	Flags	Services	Regexp	Replacement
10	0	"S"	"SIP+D2U"	""	_sip._udp.isoc.nl.

The Order and Preference denote which Registrars should be tried first to contact. As our query returned only one record, that record is the one to be followed. The Flags field indicates that this NAPTR record points us to a SRV record. The Services field indicates that we can contact that server over UDP. The Services field is either `"SIP+D2`*X*`"` or `"SIPS+D2`*X*`"`, where *X* denotes the underlying transport protocol: `U` for UDP, `T` for TCP and `S` for SCTP. The Regexp and Replacement fields should be interpreted as: replace the path in the SIP address that matches the regular expression given by Regexp (in our case the whole address), and replace it with Replacement `_sip._udp.isoc.nl.`.
We now have to do a SRV record[5] lookup of `_sip._udp.isoc.nl`. This gives at the time of writing: `0 0 5060 sip.dns-hosting.info.`. The fields

in this returned value are:

Priority	Weight	Port	Target
0	0	5060	`sip.dns-hosting.info.`

The first two fields, Priority and Weight, again denote the preference in which to try the returned records. Since we have only one record, this does not apply. The Target and Port specify which host to contact, on which port. A host and port can be transformed in a contact-address. The contact-address of the Registrar for `isoc.nl` is thus: `sip:sip.dns-hosting.nl:5060;transport=udp`.

2.2.2 Authenticating to the registrar

The registrar is identified to the UA by a TLS certificate in case of SIPS[3], or not at all, in case of SIP. The registrar authenticates the user with HTTP Digest Authentication[7].

After a registration request is made by an unauthenticated UA, the Registrar first send the UA a challenge (nonce). The UA responds by returning a MD5 hash of two other MD5 hashes and the nonce. One of the rehashed MD5 hashes is a hash of the end-users username, password and the realm (the domain part of the contact-address of the Registrar). The other rehashed MD5 is a hash of the method (REGISTER) and the URI of the request.

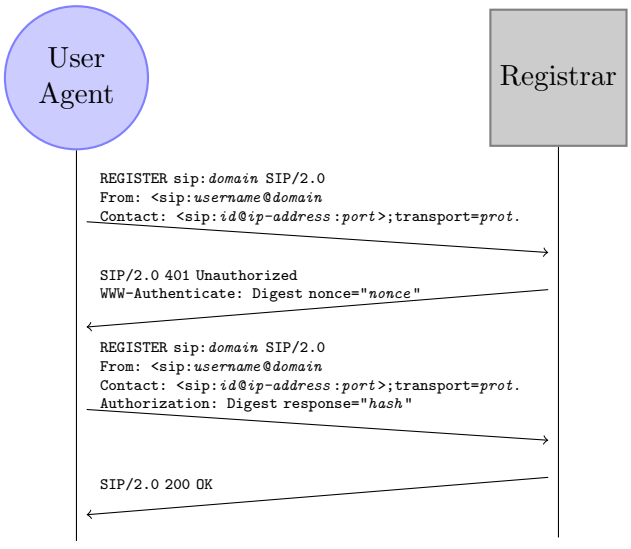


Figure 1: Registration

In figure 1, such a transaction is visualized. SIP messages look like those defined in HTTP[8]. They follow a request/response transaction model.

[3]SIPS is SIP with TLS[6]

A request consists of a text line containing the method, the arguments for the method and a protocol specifier (`SIP/2.0` in case of SIP). After that a set of headers, followed by an empty line and an optional message payload follows the Internet Message Format[9] standard. The first line of a response consists of a protocol specifier, followed by a status code (à la Enhanced Mail System Status Codes[10]) and a textual representation of the status.

The method used to register an UA is: `REGISTER`. The address-of-record to bind a contact-address for, is in the `From:` header. The contact-address, is in the `Contact:` header. The Registrar should store this binding in a Location service (a database) which is also accessible for the SIP network elements that need to lookup the contact-address.

2.3 Calling out

After registration an end-user can initiate a session with another end-user by using the address-of-record for this end-user. The UA, acting on behalf of that end-user, does this by sending a `INVITE` request to the SIP Proxy for the address-of-record. A UA finds the Proxy using DNS or a by sending to the "all SIP servers" multicast address: `sip.mcast.net` (`224.0.1.75`), in the same manner Registrars can be found (see 2.2.1).

Note that as there is no difference in DNS for an entry for a Registrar and Proxy, when a UA finds its Registrar via DNS, it is also the UA's Proxy. If the Registrar and the Proxy for a certain domain are on different hosts, the Registrar has to be configured manually for the UA using address-of-records for that domain. Note that an UA does not have to know the proxy for its own domain. It is only interested in contacting the proxy for the address-of-record it wishes to reach.

In figure 2 is shown how the UAs and the Proxy interact in setting up a session. The encircled numbers indicate the steps involved.

1. Alice's UA sends a `INVITE` request to Bob's Proxy. Included in the request is a description of the type of session Alice wishes to initiate. It is described using SDP, the Session Description Protocol[11].

   After the Proxy received the invitation from Alice, it first confirms that it will try to contact Bob for her, by replying with a `100 Trying` status response. It then looks up the contact-address for Bob, and sends the `INVITE` request to that address.

2. Bob's UA received Alice's invitation. If it can handle the media described in the SDP, it informs Bob (for example with a ringing sound). It then informs the Proxy that is has informed Bob about the request. The Proxy forwards this to Alice's UA.

3. Bob has heard Alice's invitation and decides to answer the call. The UA sets up a listening socket to accept the session on. It then con-

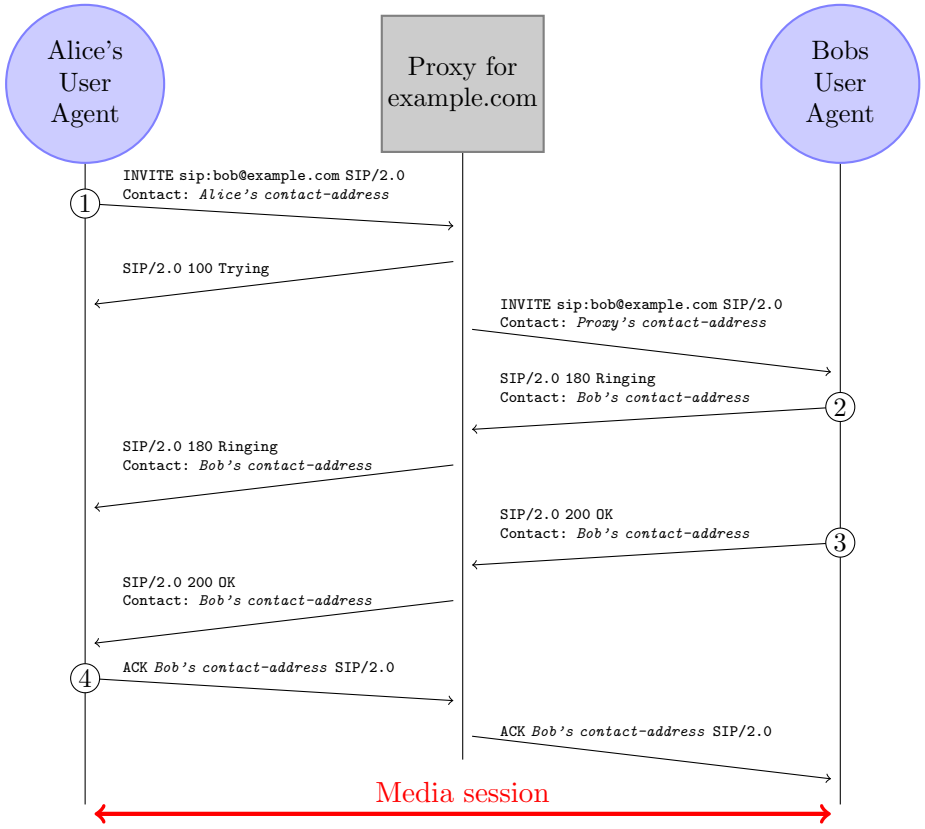


Figure 2: Making a call

structs a new SDP message not only confirming the type of media, but also providing the listening socket's IP-address, port and protocol. Alice's UA may contact that to initiate the session. It sends this SDP with a `200 OK` status response to the Proxy. The Proxy forwards it to Alice's UA.

4. Alice's UA confirms that it will setup the session by sending Bob's proxy a `ACK` method. The Proxy forwards this to Bob's UA. Alice's UA initiates the media session directly with the protocol and to the IP-address and port that Bob's UA provided in the SDP in step 3.

2.3.1 The SDP format

SDP is used to agree upon the media type and format to be used, and the host, port and protocol on which to setup the media session. It does this with a request/response transaction model. The request indicates the media type and an optional list of supported encoding formats by the requesting party. The response then narrows the list of supported encoding formats to

the formats it supports itself, and mentions an IP-address, port and protocol that can be used by the requesting party to initiate the media session. The requesting party then picks its favourite encoding format and initiates the media session.

```
v=0
o=- 2713 2713 IN IP4 145.100.104.30
c=IN IP4 145.100.104.30
m=audio 51942 RTP/AVP 104 103 102 101
a=rtcp:51943 IN IP4 145.100.104.30
```

SDP Request

```
v=0
o=- 2766 2767 IN IP4 192.0.0.33
c=IN IP4 192.0.0.33
m=audio 40036 RTP/AVP 104 101
a=rtcp:40037 IN IP4 192.0.0.33
```

SDP Response

Figure 3: A SDP Request & Response message

In figure 3 a stripped form of a SDP request and response is presented. A SDP message consists of a list of *field=value* pairs. The fields are designated by a single character.

The fields have the following meaning:

v gives the version number and is always "0".

o gives the originator of the session. The value is a list of values. The values of the request were:

username	sess-id	sess-version	nettype	addrtype	unicast-address
-	2713	2713	IN	IP4	145.100.104.30

username Not applicable in this media session negotiation. SIP takes care of that.

sess-id Must together with the remaining values, except for *sess-version* be unique for this session negotiation.

sess-version Its usage is up to the creating tool. But it should be increased every time the SDP message is modified.

nettype The type of network. Currently only `IN` is defined to have the meaning "Internet."

addrtype The type of address. Currently only `IP4` and `IP6` are defined.

unicast-address The address of the machine from which the session was created.

c contains the connection data. It is a list of values for *nettype*, *addrtype* and *unicast-address* which have the same meaning as in the *originator* field.

m This line give the media type and formats requested and agreed upon. It is a list of values. The values of the request were:

media	port	protocol	format
audio	51942	RTP/AVP	104 103 102 101

media The media type. Defined media types are: `audio`, `video`, `text`, `application` and `message`. It reflects the media-types list maintained by IANA (See http://www.iana.org/assignments/media-types/), and more can be found there.

port The transport port to which the media stream is sent.

protocol The protocols used to transport the media stream. It can be layered. The layers are divided by slashes. In our example `RTP/AVP` means: The `AVP` protocol over `RTP`. `RTP` implies that the transport layer protocol is `UDP`. Other possible values are listed bye IANA here: http://www.iana.org/assignments/sdp-parameters. The list includes the "TCP/TLS/MSRP" protocol (`MSRP` over `TLS` over `TCP`) that we will explore later in this report.

format A protocol specific list of formats for the protocol. In a request, a list is presented that the requester supports. In the response this is narrow down to also match the formats the called party supports. The requester then eventually picks its favourite format from that list.

a contain optional extra attributes for the media defined in the media field. In our example an address is given to control the RTP media stream.

2.4 Redirections

With SIP it is possible to redirect calls for an address-of-record to a different URL. SIP doesn't specify how this redirection of an address should be registered; This is left for the creators of SIP network elements to decide. It only specifies the transactions involved to follow a redirection. For that purpose it introduces a special network element: the Redirect server.

In figure 4 an example is given of the role of a Redirect server in a redirection transaction. Alice sends a `INVITE` request to the Proxy of the example.com domain. This accidentally happens also to be a Redirect server, and it informs Alice's UA by responding with a "`3`" class response code (redirection). The URL that Alice's UA should follow is given in the `Contact` header. Alice's UA confirms the reception of the redirect by replying with an `ACK`.

Alice's UA can now inform Alice about the redirection, and ask if Alice wants to follow the given URL. Or it could just follow the given URL. This is up to the creators of the UAs.

When the URL is followed and, as in our example, is a SIP address-of-record, the invitation is then continued as described in subsection 2.3.

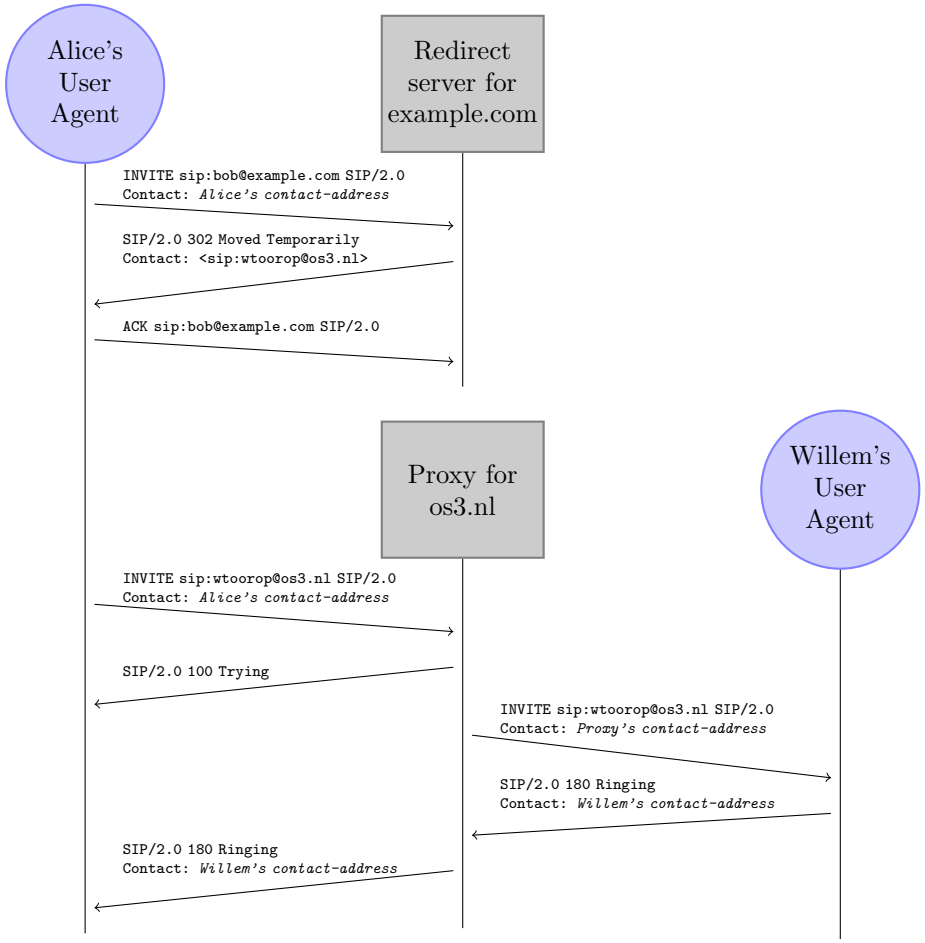


Figure 4: Redirection

2.5 Other features of SIP

Other features, usually seen in telephony networks, such as conference calls, are not defined in the SIP specifications. It is up to the implementors of User Agents to make them work. Often such a feature is highly dependent on the media type as well.

For example, with a conference call, for audio streams a participant would like to have the audio streams generated by all other participants mixed in a single audio stream. But the video streams should not be mixed, and presented as a separate video channels that show all other participants. In the context of Instant Messaging, a conference call (or a call to a chat-room), means that the messages of the participants are presented sequential after each other.

SIP is only responsible for the initiation of the sessions, and leaves all

such issues with the media-type specifications and the implementors of the UAs.

3 The NAT-Traversal problem

In this section is explained: what NAT is, what its advantages are, and what the problem is with NAT: (The NAT-Traversal problem). Then the extensions for SIP to deal the NAT-Traversal problem are explained.

The media sessions also have to deal with the NAT-Traversal problem. First the Industry provided solution types for the problem are given. Finally it sums up the solutions the IETF came up with.

3.1 What is NAT?

Network Address Translation (NAT) is a technique with which the network address information is altered in a packet when it passes through a router.

Most common usage is network masquerading: the (private) source addresses of a network are altered to a single (public) address. With this technique an internet connection which services a single IP-address (as is common with consumer ADSL and cable subscriptions), can have multiple computers use that internet connection. Other forms of NAT include DNAT, or Destination NAT, which alters the destination address of a packet crossing a router, but they are not of relevance for our research and not further explained or explored in this report.

NAT has a widespread usage on the internet today. ADSL and Cable-modem users are almost guaranteed, that their modem masquerades their internal IP-address to a single public outside IP-address. Another advantage using Masquerading NAT for corporates is that they don't have to change their internal IP-addressing plan if the switch internet provider.

It also, as a side effect, protects the computers on the inside site of the router against incoming connections from the outside (the internet). The cause for this side effect is explained in the next subsection 3.2. This causes problems for protocols such as SIP, because a User Agent on the inside can not be contacted by the Proxy for the domain it has an address-of-record for. Also the media session can not be setup between end-points, because it will be an incoming connection from the outside for one of the participants. This is known as the NAT-Traversal problem.

3.2 How does NAT work?

Figure 5 shows a picture of a typical NAT setup. The *Client* on the inside site of the router connects through the NAT router to a *Server*. The NAT router then takes the source IP-address and source port of the initial connection packet, and looks if there is an entry for the pair in the *Internal* column of

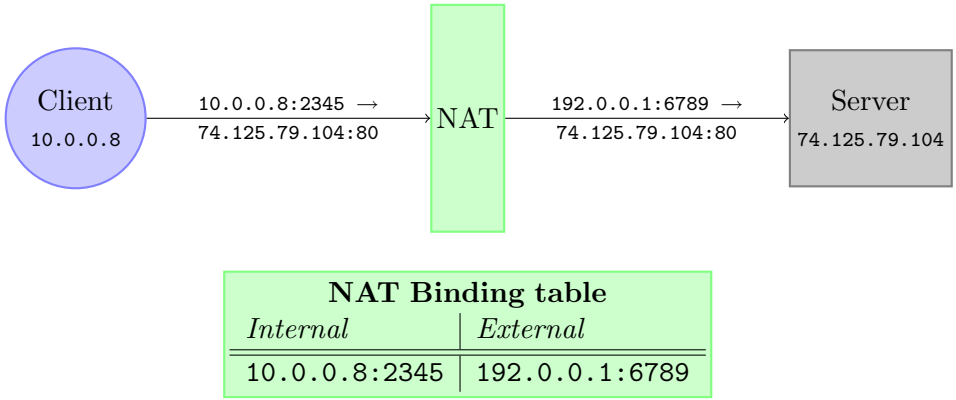


Figure 5: Network Address Translation

its *NAT Binding table*. If there is an entry it replaces the source IP-address and source port of the packet with the values in the *External* column and forwards it to the *Server* on the outside. If there isn't an entry, it generates a port that is not yet in use in the *External* column of the binding table, and stores an new *Internal* address:port, *External* address:port in its binding table.

Now when a response is received from the *Server*, it will be targeted to the NAT router itself. The NAT router then has to lookup the port number, the packet was targeted towards in the *External* column of its binding table, and translate the destination IP-address and port to the values in the *Internal* column on the same row.

When no incoming packets have been received, either from the inside or outside, for a certain amount of time for a row in the binding table, the row is removed from the table.

Besides the fact that the *Client* has an IP-address from the Private Address Space[12], the client cannot be connected because the NAT router has to find a matching *External* entry in its binding table for all incoming packets. The only way to reach a host on the inside of a NAT, is for that host to make an entry in the NAT binding table by sending a packet first. The establishment of a connection to the inside of a network behind a NAT is called *NAT-Traversal*. The unreachability of a host "behind" a NAT without having sent an initial packet is called the *NAT-Traversal problem*.

3.3 How does SIP deal with NAT?

From the IETF, two reports describing how to deal with SIP and NAT have emerged. The first is RFC 3581[13]: "An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing." The extension specifies an extra argument in the `Via` header of a SIP message; namely `rport`. The `Via` header is used to indicate the transport used for the transaction and

identifies the location where the response is to be sent. The `rport` parameter from the RFC allows a client to request that the server sends the response back to the source IP address and port where the request came from. Before the RFC, a port specified in the message itself was used (when using TCP transport), or the default SIP port 5060.

The second report, still in draft, but in widespread use today, is draft-ietf-sip-outbound-16[14]: "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)." It defines the usage of a Proxy acting on behalf of the User Agent. This was also a possibility defined in the SIP specification, but the draft extends it by specifying that the initial UA initiated connection should be reused to send messages back from the Proxy to the UA. It also gives a method for keeping the bindings for that initial connection in the NAT binding table, by regularly generating "Keep Alive" messages.

In figure 6 is demonstrated how Alice's UA can be reached through her *Outbound Proxy*. The red line though the NAT is indicating that she initially made the connection on which Bob's `INVITE` is received.

Note that in figure 6, the Outbound Proxy and the regular SIP Proxy for `exmample.com` are two different hosts. This does not have to be so. The Outbound Proxy is another SIP network element, and may be integrated with a Registrar, Proxy, Redirect server in one single host.

3.4 What about the sessions themselves?

The two IETF reports solve the NAT-Traversal problem for SIP, but it does nothing for the sessions initiated through SIP.

3.4.1 Industry answers

The market first addressed the issue by supplying so called Application-level gateways (ALG). An ALG sits on the NAT, and does deep packet inspection and alteration of SIP messages and the SDP therein. When a UA behind a ALG-NAT is accepting an invitation of another UA, the following steps are followed to ensure that the NAT for the media session in traversed:

- The UA opens listening sockets for the incoming media stream.
- The UA sents a SDP response mentioning the listening sockets and the media type and format on which it agreed upon.
- The SIP message containing the SDP response enters the ALG-NAT.
- The ALG chooses a free port for each connection specifier in the SDP
- It updates its NAT binding table with the chosen ports in the *External* column and the addresses, ports and protocols of the matching connection specifiers in the *Internal* column.

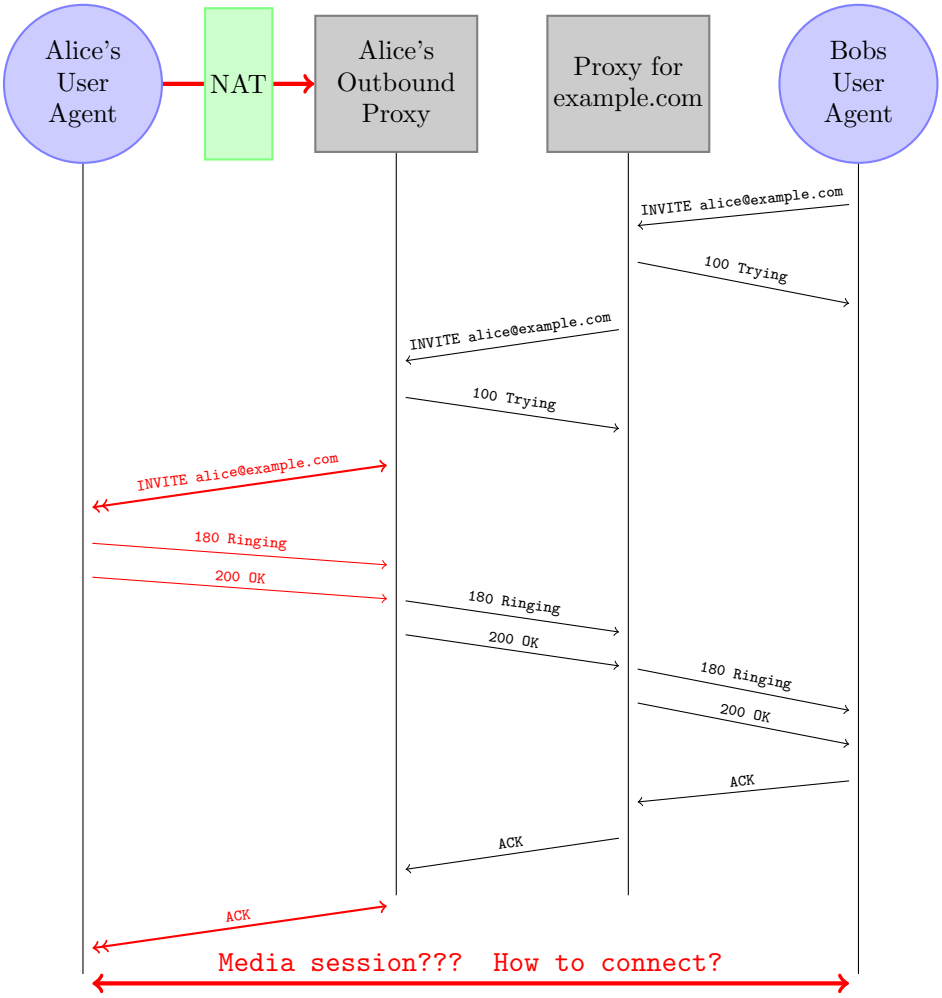


Figure 6: Edge/Outbound Proxy

- It then rewrites the SDP to contain its own public IP-address, with the ports it just has chosen.
- It then sends out the SIP message.

Initiations for the media stream will now connect with the ALG-NAT, which forward it to the ports the UA reserved for the stream.

This solution has the following disadvantages:

1. It fiddles with SIP's content, which makes problem solving less transparent
2. It does not work with SIP over TLS.
3. It has to understand SDP, which evolves:

Figure 7: Application-level gateway

Attributes containing connectible addresses could be defined after the ALG was programmed. It thus has intelligence in the infrastructure that actually has to be in the User Agents.

4. It requires special NAT equipment

Especially because of the fourth, it will never have widespread usage. Because of these disadvantages the market came with another solution: the Session Border Controller (SBC).

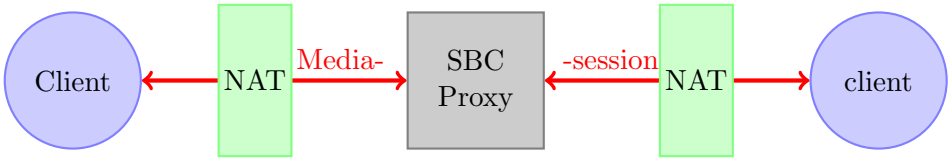


Figure 8: Session Border Controller

A SBC behaviour is similar to that of an ALG, it inspects SIP messages and alters the containing SDP to contain connection specifiers of a relay. It asks both parties involved via the modified SDPs to connect to the relay. In figure 8 the SBC is the relay itself.

SBCs are easily deployed and do not require alternate network equipment. This makes them very popular. SBCs are currently widespread deployed.

They do however inherit some of the disadvantages of ALGs: They make SIP traffic harder to debug, but most importantly, they break the concept of having all session intelligence on the User Agents, which reduces SIPs extensibility.

3.4.2 The IETF answer

The IETF has developed several methods to deal with NAT-Traversal. The first they came up with is STUN, which exploits the way NAT equipment operates. In some cases it is still possible to initiate connection through a client behind a NAT, although this only applies to stateless transport protocols (UDP!). STUN helps discovering the type of NAT a client is behind so it can determine if such a case is possible.

The second solution defined by the IETF is TURN, which is an extension of STUN. TURN is a STUN server that can operate as a relay if no other options are available. TURN can thus also support statefull transport protocols (such as TCP), but that is expensive, because all traffic of the media session in such a case is relayed through the TURN server.

Finally ICE and ICE-TCP define how SIP User Agents should make use of STUN and TURN servers, and mention all the available connectivity from them in SDP messages, with which they may agree upon how the media session connection should be made.

3.4.2.1 STUN Session Traversal Utilities for NAT[15]

STUN is a lightweight protocol which can help clients, to discover if they are behind a NAT and if so, what type it is. The type of NAT is the way a NAT router operates. Four different types of NAT were defined in the STUN RFC 3489[16]: Full cone NAT, Address restricted cone NAT, Port restricted cone NAT and Symmetric NAT. Although the new RFC 5389 does not mention these types explicitly, I find it useful to examine those types, to learn how STUN can help with NAT-Traversal. The basic operation of STUN which we cover in this section, has not been updated by the new RFC.

First we will explain how each different NAT operates. For every NAT type, the possible options for setting up a media session are covered. Then is explained how a STUN server can help in the determination of the NAT type.

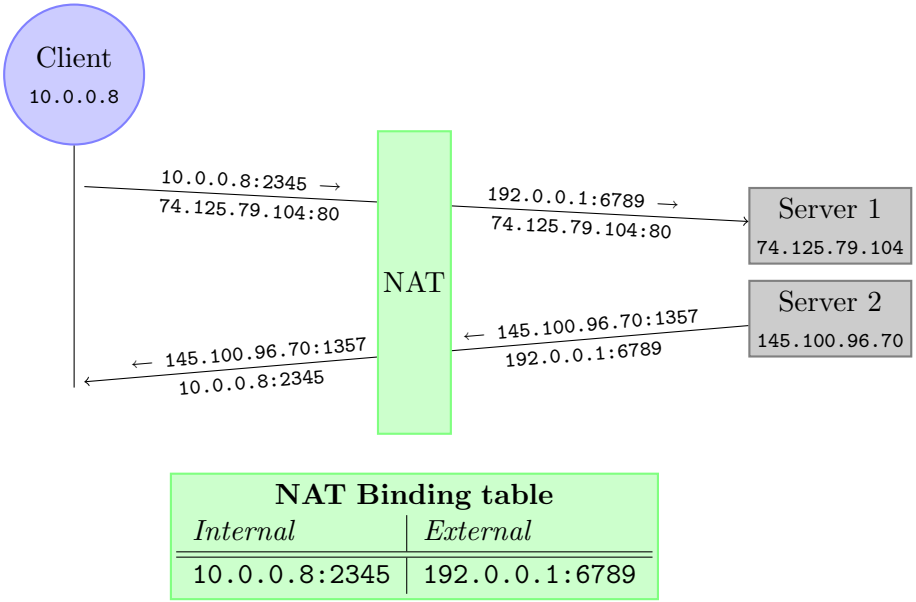
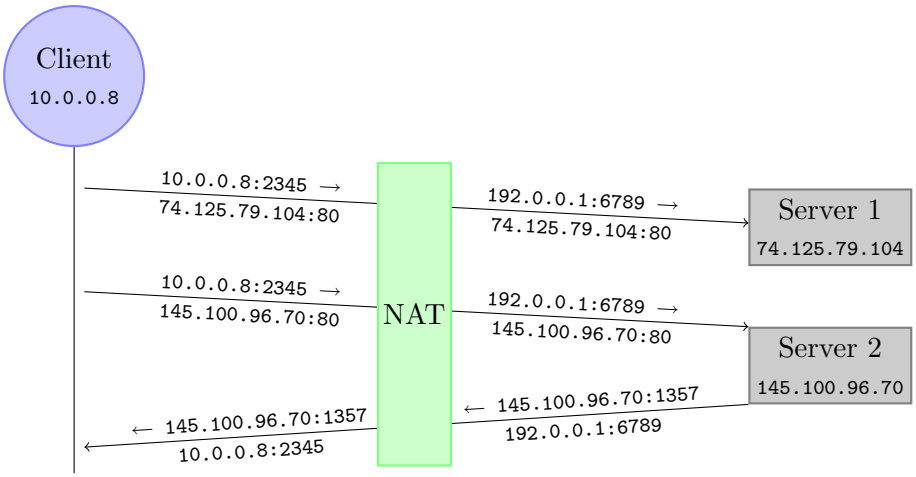


Figure 9: Full cone NAT

In figure 9 is visualized what is allowed in the operation of a Full cone NAT. A Full cone NAT, only stores an internal source IP-address and port number, and the IP-address and port number it was translated to in its binding table. If it only stores this data, the same external IP-address and port will be used for connections to other external IP-addresses, as long as the originating client uses the same source port. Furthermore, any packet targeted at the external IP-address and port number will therefor be forwarded by the NAT to the inside.

This is very good, because the inside hosts are basically connectible from the outside on a specific port, as long as it has sent a packet through the NAT originating from that port.



NAT Binding table

Internal	*External*	*Servers*
10.0.0.8:2345	192.0.0.1:6789	74.125.79.104 145.100.96.70

Figure 10: Address restricted cone NAT

Figure 10 shows the operation of an Address restricted cone NAT. Such a NAT also stores what servers have been contacted from a inside source address/port pair. Only servers that the client has sent a packet to from a specific port, can send packets to the client by using its external address/port pair. The destination port the packet was sent to from the client is not stored in the binding table. This means that the server can sent to the client from any source port it likes.

This is also good for setting up media sessions. The client that wants to open a listening socket on a specific port, just has to send the party that will connect a packet that originated from that port. It does not matter if the other party does not receive this packet when it is itself behind a NAT,

because we know what external port to contact to reply and can specify that port in the SDP. However, we do in such a case have to know the external IP-address of the other party. The other party can discover its own external IP-address using STUN, and inform us about it in the initial SDP.

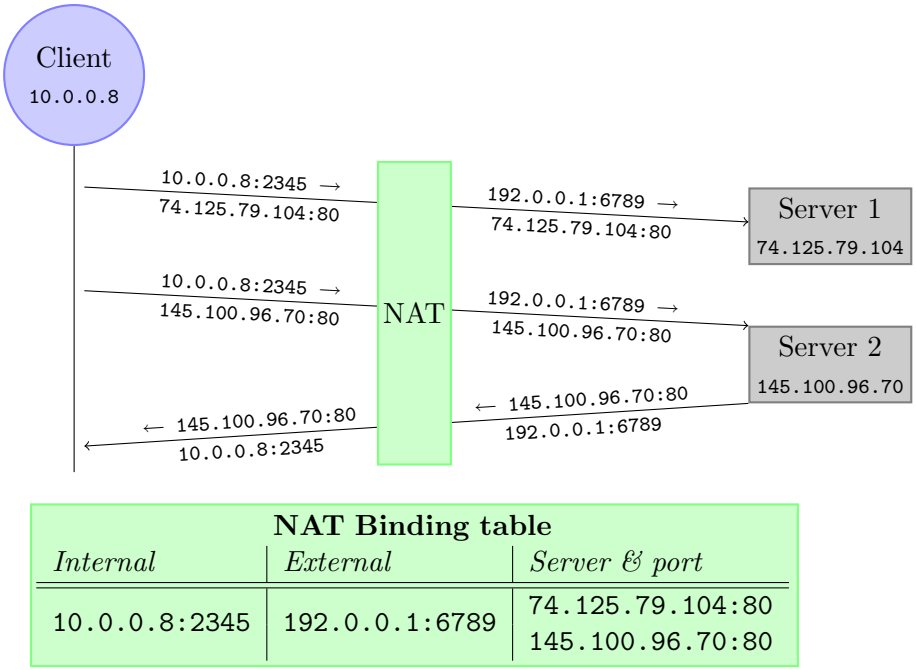


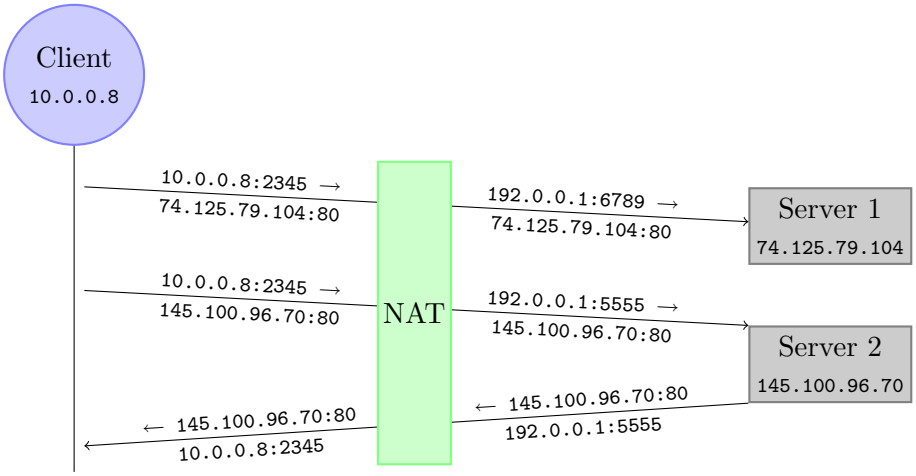
Figure 11: Port restricted cone NAT

Port restricted cone NAT as shown in figure 11 is much like Address restricted cone NAT, but now also the port of the server that was sent a packet to is stored in the binding table. A packet is only forwarded to the inside client, if the packet originated from the IP-address and port where the client first sent its packet to.

With such a NAT we are still able to setup a media session. The initiating party has to choose a port and discover to which external port it is translated by the NAT by contacting a STUN server. The called party does the same. When agreed upon the session they just start sending each other messages from the same source port, to the others party external addresses and ports that where discovered using the STUN server. Each party is able to receive a packet on that port, once it has sent an initial packet to the other party.

With Symmetric NAT, figure 12 each external IP-address/port pair has its own entry in the binding table, and is translated to different source ports. Only a server that received a packet from the inside client can see to which external port it is translated.

When two clients are both behind a Symmetric NAT, no media session can be setup. Direct media session connections between two parties for



NAT Binding table

Internal from	*External to*	*External from*
10.0.0.8:2345	74.125.79.104:80	192.0.0.1:6789
10.0.0.8:2345	145.100.96.70:80	192.0.0.1:5555

Figure 12: Symmetric NAT

which one is behind a Symmetric NAT can be setup if:

1. The Symmetric NAT translates to the same external IP-address, and the other party is beheind Address restricted cone NAT.

   When behind Address restricted NAT, a port can be opened for the external address by sending a packet from that port to the external IP-address of the client behind the Symmetric NAT. Port restricted cone NAT would not work, because we can not know in advance what port the client behind Symmetric NAT will use to connect.

2. The other party is behind Full cone NAT.

   The port the client detected with STUN is also usable for incoming traffic from any host.

3. The other party is not behind a NAT and not behind a firewall.

A STUN server has two different IP-addresses on the public internet. A client can send a binding request to a STUN server. The STUN server copies the IP-address and port it saw when receiving the packet in the response. The client compares this with the local IP-address and port it bound to when it sent the request. If the values differ, it knows it is behind a NAT.

It can then send new binding requests to the STUN server in which it can ask the STUN server to reply with a different source port, a different

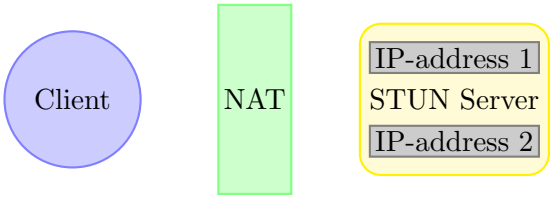


Figure 13: STUN Server

source IP-address, or both. It then waits for a certain time (9.5 seconds) for a reply. If it doesn't receive a response within that period, it learned that its NAT does not support the requested operation.

With a properly defined flowchart (as in RFC 3489) the client can determine behind what type of NAT it is.

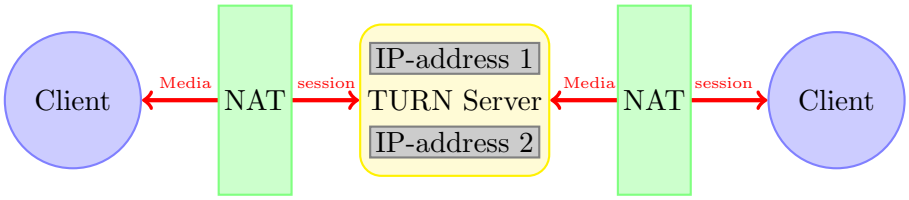


Figure 14: TURN Server

3.4.2.2 TURN Traversal Using Relays around NAT: Relay Extensions to Session Traversal Utilities for NAT.

TURN is not a settled standard yet. As the title implies, the draft, draft-ietf-behave-turn-12[17], specifies an extension to STUN, in which clients can use the STUN (or now TURN) server to relay data streams for them.

3.4.2.3 ICE Interactive Connectivity Establishment: A Protocol for Network Address Translator Traversal for Offer/Answer Protocols.

ICE is also not a settled standard yet. ICE defines a procedure that SIP User Agents should follow, which give them the best and shortest media session using UDP. ICE does it by defining nine steps:

1. **Allocation** The UA that wants to place a call gathers its candidates for media stream connections using STUN and TURN.

2. **Prioritization** The candidates are sorted. Direct IP-addresses and ports have the highest priority. External NAT addresses follow, and relayed TURN addresses have the lowest priority.

3. **Initiation** The candidates with their priority are packed in SDP and sent to the called party with the `INVITE` request.

4. **Allocation** The called party now does the same as the caller in step 1 and 2.

5. **Information** The called party does not ring the phone. It sends a provisional message containing its candidates to the caller.

6. **Verification** The caller and the callee now start sending messages to and from all the candidates. They sort all the successful connection pairs on preferable priority.

7. **Coordination** The candidates are now *all* tested in priority order. This is done by STUN requests to each other on the connections to be used for the media streams. This allows for delays in higher priority candidates due to packet loss, and allows other criteria such as Round Trip Time.

8. **Communication** Media can now flow in both directions. The caller can, for example, already show himself to the callee.

9. **Confirmation** A re-invitation is sent, with the selected candidate-pairs in the SDP. Thereby all the SIP network elements involved know which media paths are actually going to be used.

3.4.2.4 ICE-TCP TCP Candidates with Interactive Connectivity Establishment

This is a draft that extents the procedure described in ICE to be used with TCP.

3.4.3 Media specific solutions

So far, we've seen general solutions for NAT-Traversal for media sessions. They were also mostly oriented towards UDP based media streams. It is also possible to provide a solution for a specific media type. One such solution is MSRP-Relays[18] which is an IETF standard for a proxy for the Message Session Relay Protocol[19].

3.4.3.1 MSRP The Message Session Relay Protocol.

MSRP is a media type for instant messaging int the context of a session. This is different from the other well defined messaging standard in SIP, which is page-mode messaging as defined in RFC 3428[20]: "Session Initiation Protocol (SIP) Extension for Instant Messaging." Page-mode messaging uses SIP signaling to deliver the message by extending SIP with a `MESSAGE` method . It operates just like a text-message set with a cell-phone. Session-mode instant messaging uses the conventional scheme of SIP session setup. The message type to be agreed upon in the SDP is then `message`.

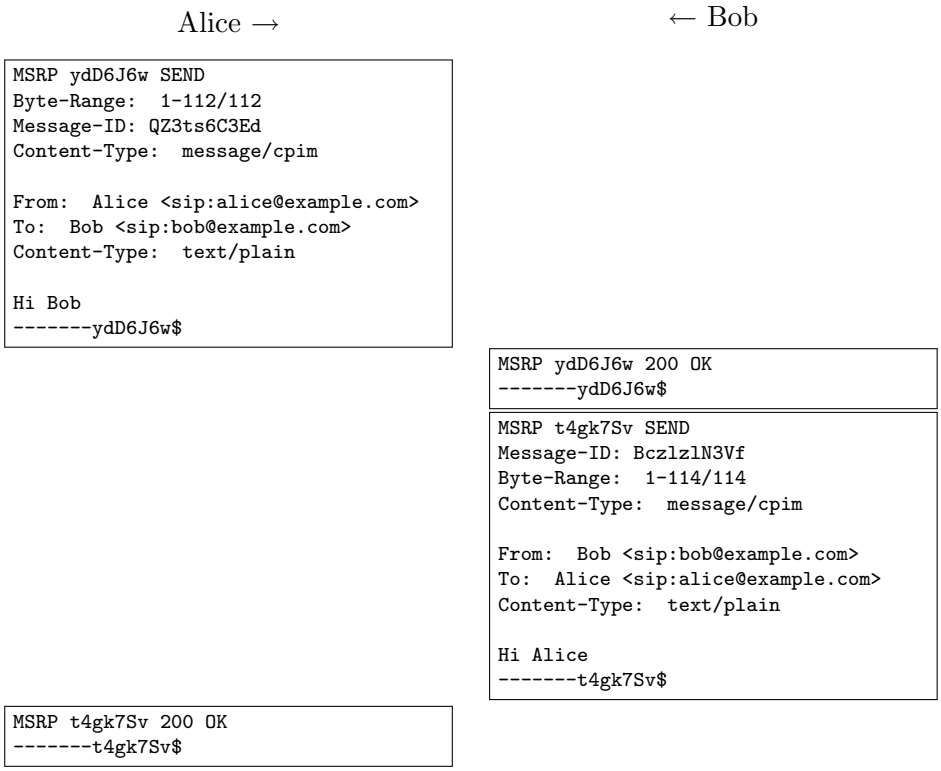


Figure 15: A MSRP request/response transaction

MSRP has many aspects in common with SIP and HTTP. It also uses a request/response transaction model. Its requests are also formated as a first line containing a request method, and the remainder of the message as in the Internet Message Format. Only, with MSRP, the requests and responses can come from two sides. Also, the underlying transport TCP is not closed after the response has been received, but remains open until by SIP signaling is received that the session is terminated. This means that extra attention has to be given to mark the beginning end ending of a MSRP message.

In figure 17 we see an example of a possible MSRP transaction. The messages are a bit simplified. Especially there are no `To-Path` and `From-Path` headers, which are always part of a MSRP message, and contain the URL for the local and remote MSRP services.

Note that the MSRP messages have a `Content-Type: message/cpim` header/value pair. The `message/cpim` format[21] itself is an format using the Internet Message Format. In this example `message/cpim` serves no other purpose that to explicitly name the sender of the message. In a session with only two participants this might not be necessary, but with a conference instant messaging session (a chatroom) this information is very useful.

With the `Content-Type` header, MSRP can be considered a transport

protocol for other media types itself. And indeed the SDP messages used to agree upon a MSRP session reflect this.

```
v=0
o=- 2713 2713 IN IP4 145.100.104.30
c=IN IP4 145.100.104.30
m=message 12345 TCP/TLS/MSRP *
a=accept-types:text/plain message/cpim *
a=accept-wrapped-types:text/plain
```

SDP Request

```
v=0
o=- 2766 2767 IN IP4 192.0.0.33
c=IN IP4 192.0.0.33
m=message 8888 TCP/TLS/MSRP *
a=accept-types:text/plain message/cpim *
a=accept-wrapped-types:text/plain
```

SDP Response

Figure 16: SDP messages for a MSRP session

In figure 16 a SDP request/response pair is given for a MSRP session. The protocol specifier in the media field is defined as `MSRP` over `TLS` over `TCP`. The media types to be transported over MSRP is defined in the attribute fields. The `accept-types` attribute contains a list of the media-types that may be transported over this particular MSRP session. Since the `message/cpim` media type is acceptable, an extra attribute is present specifying what media-types are acceptable within a `message/cpim` message: `accept-wrapped-types`

The asterisk at the end of the `accept-types` attribute, denotes that the other party may attempt to send content with media-types that have not been explicitly listed. This can be used to allow file transfers over MSRP as defined in draft-ietf-mmusic-file-transfer-mech-10[22].

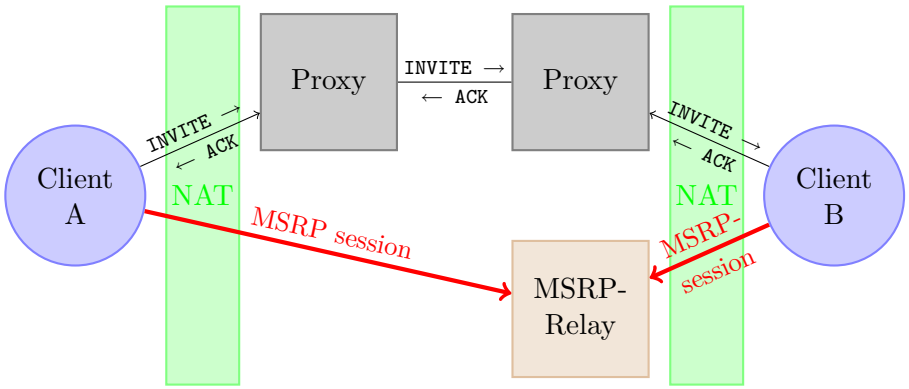


Figure 17: Message Session Relay Protocol

3.4.3.2 MSRP-Relays A MSRP-Relay defines a proxy for MSRP. It can be used as follows:

1. When a invitation for a MSRP session is received, the called party authorizes to the MSRP-Relay. A relay can either be configured manually, or be discovered by DNS. Servers can by found by either querying

a SRV record for _msrps._tcp.*domain* or _msrp._tcp.*domain*, which refer to MSRP servers respectively using TLS and plain TCP.

2. In the success response of the authentication process is a *Use-Path* header included. The value of that header is an MSRP URL for the caller to connect to. The connection to the relay is kept open.

3. The called party responds to the invitation with an SDP that has an *path* attribute field which has as its value, the value just returned by the relay in the *Use-Path* header.

4. The caller can now connects to the MSRP-Relay with the URL it found in the *path* attribute field in the SDP.

5. The callee receives messages sent by the caller through the open connecting to the relay.

3.5 Evaluation

In this section we have seen how the NAT-Traversal problem is handled by SIP. SIP does however not help with NAT-Traversal for the media session.

Industry has found a working solution, SBC, that is in widespread use. SBCs however have the disadvantage that they put knowledge about the media sessions in the SIP network elements, which reduces SIPs extensibility.

The IETF has come up with a solution that is relatively new and still a draft and not a settled standard; ICE and ICE-TCP. Because of its immaturity, it is not widely deployed. Besides that, it solves issues that are addressed already by SBCs. It also requires the availability of STUN and TURN servers, that have the peculiar property that they need to operate from two IP-addresses from the same server.

For instant messaging there is a solution for NAT-Traversal that is a settled standard: MSRP-Relays. Because of its maturity and its relative simplicity, SIP infrastructure supporting instant messaging is very likely to have a MSRP-Relay server. We have seen that MSRP is a transport protocol in itself and can be used to carry anything.

4 How to make it work

In this section we will explore which work has already been done by the IETF members for application sharing with SIP. We will then explore existing application and desktop sharing software and look how they fit in the SIP scheme. We will then present a solution that works in existing SIP infrastructure: RFB over MSRP. The remainder of the section covers the aspects of that solution that should be taken into account with an implementation.

4.1 IETF proposals

In the past there have been many proposals for application sharing with SIP by the IETF. I have found only one still active: draft-boyaci-avt-app-sharing-00: "RTP Payload format for Application and Desktop Sharing"[23]

In this draft defines a two new protocols for application sharing. One is called remoting and is used to send display update information. The other is called hip (Human Interface Protocol) and describes the message types to send mouse and keyboard events. The protocols look a lot like the messages generated by the RFB protocol.

The protocols are proposed as a payload format for RTP. That has the advantage that nothing has to be changed to make it work with SIP. It can work right out of the box, in existing infrastructure supporting NAT-Traversal for RTP. It does mention an optional RTP over TCP transport which wouldn't work in existing infrastructure.

One draft that has expired, but has an interesting approach is: draft-garcia-mmusic-sdp-collaboration-00: "Session Description Protocol Extensions and Conventions for Collaboration Applications"[24]

It proposes the format of SDP messages to be used for setting up sessions between several application sharing protocols, notablyRFB, T.120 and a scheme for Co-Web Browsing. The fact that is defines the SDP for so many, not very related, protocols is one of the main reasons this draft is rejected. Especially the Co-Web Browsing support was not considered a needed feature. Also T.120 was considered to complicated to deal with.

```
v=0
o=alice 4526 4526 IN IP4 145.100.104.30
c=IN IP4 145.100.104.30
m=application 42034 TCP/RFB *
a=setup:active
a=connection:new
```

SDP Request

```
v=0
o=bob 4527 4527 IN IP4 192.0.0.33
c=IN IP4 192.0.0.33
m=application 5900 TCP/RFB *
a=setup:passive
a=connection:new
```

SDP Response

Figure 18: SDP messages used for RFB over TCP

Figure 18 shows the proposal for the SDP for a RFB session in draft-garcia-mmusic-sdp-collaboration-00. Two attribute fields are given that are defined in RFC 4145[25] "TCP-Based Media Transport in the Session Description Protocol (SDP)": `setup` and `connection`. The `connection` attribute indicates that the media session should be a new TCP connection. The `setup` attribute indicates that Alice will initiate a RFB over TCP connection to Bob.

The disadvantages of this approach are that it probably doesn't work in current SIP infrastructure. ICE and especially ICE-TCP should be common practice to be able to NAT-Traverse regular TCP based sessions. It also requires a new protocol field for the media line to be registered with IANA.

4.2 How to make it work now

RFB is a protocol that can be used for desktop sharing. An earlier attempt to specify a draft for SDP messages describing a RFB session has failed, primarily because the draft also tried to deal with other protocols (See section 4.1). It was also likely that the NAT-Traversal for the RFB session will not work in current SIP infrastructure.

In section 3.5 we saw that MSRP can be used to carry any media-type. A MSRP session is likely to be traversing the NAT, because of the well defined standards for it (MSRP-Relays). In the following sections I will investigate RFB over MSRP as a possible solution to realise desktop sharing with SIP.

A research group at Princeton University Computer Science Department have created an implementation of a RFB server, viewer and controlling application, that enables *application* sharing in 2005/2006. It has an implementation for Linux, MS-Windows and Mac OS-X. Unfortunately it implemented the Linux server as a X-windows module. It is not compatible with modern X-windows servers anymore. The module is based on the xf4vnc X-windows RFB server module, that also has seen no recent development.

I suggest an implementation of this adaptation of RFB for the x11vnc RFB server as x11vnc is not a X-Windows server module, and thus less dependent on the development of X-Windows. Also x11vnc is still actively developed.

4.2.1 RFB over MSRP

1. RFB over MSRP as a new transport protocol

```
v=0
o=- 357 357 IN IP4 192.0.0.33
c=IN IP4 192.0.0.33
m=application 2855 TCP/TLS/MSRP/RFB *
a=setup:active
a=connection:new
```

SDP Request

```
v=0
o=- 307 308 IN IP4 145.100.104.30
c=IN IP4 145.100.104.30
m=application 50488 TCP/TLS/MSRP/RFB *
a=setup:passive
a=connection:new
```

SDP Response

2. RFB as a media-type with MSRP

```
v=0
o=- 357 357 IN IP4 192.0.0.33
c=IN IP4 192.0.0.33
m=message 2855 TCP/TLS/MSRP *
a=accept-types:application/x-rfb
a=setup:active
a=connection:new
```

SDP Request

```
v=0
o=- 307 308 IN IP4 145.100.104.30
c=IN IP4 145.100.104.30
m=message 50488 TCP/TLS/MSRP *
a=accept-types:application/x-rfb
a=setup:passive
a=connection:new
```

SDP Response

Figure 19: Two scenarios of SDP for RFB over MSRP

4.2.1.1 SDP messages proposal In figure 19 two possible ways to agree upon RFB over MSRP are presented.

The first does it by specifying RFB as a protocol in the media field, much like the SDPs suggested in draft-garcia-mmusic-sdp-collaboration-00 draft. The protocol field shows exactly the session that we would like to setup: RFB over MSRP over TLS over TCP. The media-type specifier in the media field correctly indicates application. RFB is certainly not a message type media. The `setup` and `connection` attributes are borrowed from RFC 4145, just like suggested in draft-garcia-mmusic-sdp-collaboration-00 draft. The only disadvantage with this scheme is that officially all possible values for the protocol field in a media specifier should be registered with IANA.

If we don't want to go through the hassle of registering a new protocol field for media lines in SDP with IANA, we could use the second form. The SDP indicates here that we wish to agree upon an ordinary MSRP session, with application/x-rfb as he encapsulated media type. Arbitrary media subtypes may be used with MIME when they are preceded with "x-". We have to have a media-type for RFB anyway to use in the MSRP `Content-Type` header.

This might not seem to reflect precisely what we're doing, but fits well with established standards. Also draft-ietf-mmusic-file-transfer-mech-10 agrees upon file-transfer over MSRP, with a similar message type. We can still differentiate from file transfers, because of the extra `sendonly` and `recvonly` attributes that are obligatory with file transfers.

However, there is no reason why the first form would not work in practice. The SDPs are only seen by SIP network elements that do not deal with MSRP. A MSRP-Relay will never see how we agreed upon its usage and use, and the SDP has thus no influence on its operation at all. Therefor I have chosen to use the first variant in the implementation, as it reflects more clearly what we are trying to establish.

4.2.1.2 MSRP messages The MSRP messages for RFB over MSRP could look like those in figure 20. The RFB data is encapsulated directly in the MSRP message.

RFB, standard uses TCP to connect servers and viewers. In the standard scheme of RFB operation, there is no way to mix multiple viewers or servers over one TCP session. It is possible to have multiple viewers connected to the same server, viewing the same desktop, but all those viewers have there own TCP connection. Therefor there is no added value to encapsulate the RFB data in CPIM type messages.

With TCP there is no concept of packet framing. Packet framing with TCP is usually realised using request/response transaction models, but RFB does not use TCP in this way. It just treats its TCP session as a bidirectional stream. Packet framing is realised by having fixed sizes for the different

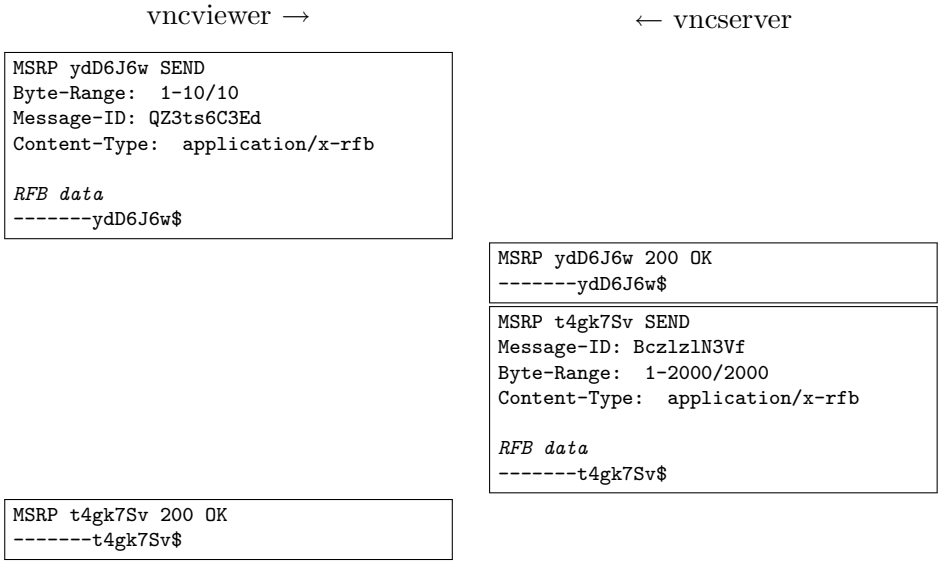


Figure 20: A RFB over MSRP request/response transaction

message types.

For our encapsulation this means that we can chunk the RFB streams in arbitrary sized pieces. The RFB software will make sure complete packets are collected from the stream before they are evaluated.

In the MSRP specification there is a small paragraph which mentions MSRP connection conserving. The idea is that a single MSRP media session is used to share all the MSRP sessions a participant has. The MSRP messages are then not encapsulated in the MSRP message itself, but they should rather be multiplexed and demultiplex by the User Agent based on the headers in the MSRP message: (particularly the `To-Path` and `From-Path` headers)

In concert with connection conserving, MSRP also defines a way to chop data into chunks. When two participants have an open MSRP session for instant messaging, and one party sends a large file to the other over that same MSRP session, the file has to be chopped into pieces which are then each sent in a MSRP message-chunk to not congest the existing instant messaging session.

None of this, is of relevance for our proposed solution. We do chunk the RFB data streams, but there is no added value in sending it in MSRP message-chunks. The RFB protocol has its own framing mechanism based on fixed sized messages and assembles the messages from the data it receives.

5 Implementation

This section covers the "proof of concept" implementation of the solution mention in 4.2. First I will describe the properties of the SIP SIMPLE client Python software library which I have used as they underlying library providing the SIP and MSRP functionality. Then we will look at available software to actually share and view a desktop over RFB. Then we will see how the implementation can connect to that software. Finally the chosen connection method is provided.

5.1 The SIP SIMPLE client Python software library

SIP SIMPLE client is an excellent easy to use Python software library to build SIP User Agents. It delivers classes and functions to easily use SIP operations, but also for media sessions such as voice, file transfer and instant messaging (including presence!).

Because it is written in Python, it should be portable to all different operating systems that can run Python scripts. This includes Linux, Mac OS-X and MS-Windows. The "proof of concept" implementation has been build on Linux.

Using the SIP SIMPLE client software it is easy to setup a listening party, or a calling party that agree upon the SDP described in section 4.2.1.1. It provides function and classes to build a console based user-interface. It also takes care of setting up the MSRP session including the optional usage of a MSRP-Relay. It can do all this and deliver an MSRPSession object, which can be used to write to and read from the MSRP session.

All this narrowed the implementation process down to two operations:

1. Read data from the RFB processing software and write it to the MSRPSession object.

2. Read data from the MSRPSession object and write it to the RFB processing software

When using an generalized interfaces for the RFB processing software and the MSRPSession object, this in itself can be narrowed down to just one function: Read data from object one and write that to object two.

The only thing remaining to be investigated is: *what interface do we have with RFB processing software to adapt to the generalized interface?* To answer this we will investigate existing RFB processing software for Linux, and then describe what interfaces could be used to use them.

5.2 The software used

RFB is originally developed by Olivetti Research Laboratory (ORL). ORL distributed a software product called "Virtual Network Computing" (VNC)

that used the protocol. The VNC product was published as Open Source, which stimulated the development of many other Open Source derivatives. The RFB protocol these products use, is not an Open standard, but is publicly available.

VNC software typically contain a server and a viewer (or vncserver and vncviewer). There are two modes of operation. The viewer connects to the server to view the desktop of the computer on which the server is running, or the server connects to the viewer providing the desktop. The second method is called "reverse connection". With both methods, the vncserver is the first to send a message. The second method has the advantage that the vncclient doesn't have to authenticate to the vncserver.

In Linux there are three types of vncservers.

1. A separate X-Windows server instance with no screen attached. This is how the original vncserver for UNIX based computers was implemented by ORL.

2. A X-Windows module.

3. A separate program using the MIT-SHM extension to access the X-Windows servers framebuffer data.

The last type is the preferable one, because it actually connects to an existing display, but is not dependent of the version of X-Windows used, as long as it supports MIT-SHM. I have found just one program of the last type: x11vnc. x11vnc also runs on Mac OS-X.

The vncviewers all operate in the same way. They do not have special requirements to access a X-display. I have found one interesting vncviewer, which was implemented in pure Python: `http://homepage.hispeed.ch/py430/python/`.

5.3 Endpoint connection types

The endpoints in a MSRP-session can connect to the vncserver and vncviewer in different ways. In this subsection an overview is given of those different ways.

The default operation of a vncserver is to listen for incomping connections. An endpoint could connect to the vncserver simply by initiating an TCP connection. Figure 21 show such an setup.

This has many disadvantages:

1. The vncserver should be running before we can connect to it.

2. We do not have the exclusive privilege of connecting to the vncserver. Other software, not even on the same computer, can connect to the server too.

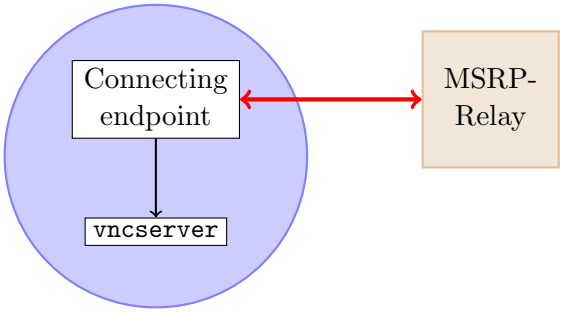


Figure 21: Connecting endpoint

3. With this modus of operation vncservers often require a password. Because authorization in our setup is handled by SIP, this is not desirable.

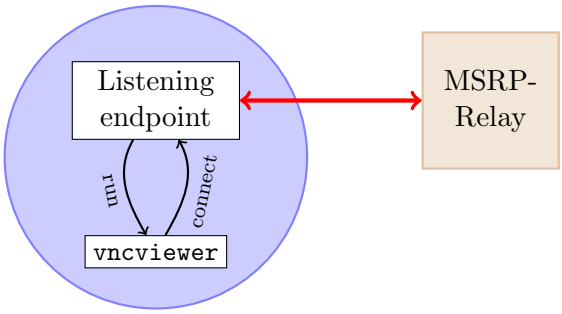


Figure 22: Listening endpoint

The default operation of a vncviewer, is to connect to a vncserver over a TCP connection. We can let the vncserver connect to a listening socket which we opened before we start the vncviewer. This setup is shown in figure 22.

None of the disadvantages mention with figure 21 applies in such a setup. We have complete control over both the TCP sessions and the program itself.

x11vnc can operate from inetd (See figure 23) It has an extra -inetd option which let it treat the standard input and output as the connection to the viewer.

Such a setup would be ideal. We have none of the disadvantages a connecting endpoint has. We even have an extra advantage over an listening endpoint in that we don't have to deal with a TCP session at all.

Unfortunately this setup did not work in practise. x11vnc expects that it is connected to through a socket. It does socket operations on the standard input/output when in -netd modus.

I have not investigated the type of operations it performs on the socket. I do not see where those would be necessary for. I suggest investigating this

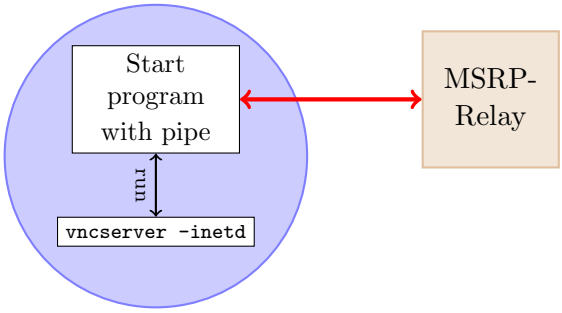


Figure 23: Piping endpoint

and providing a patch for x11vnc which removes the socket operations if they are indeed are unnecessary in inetd modus.

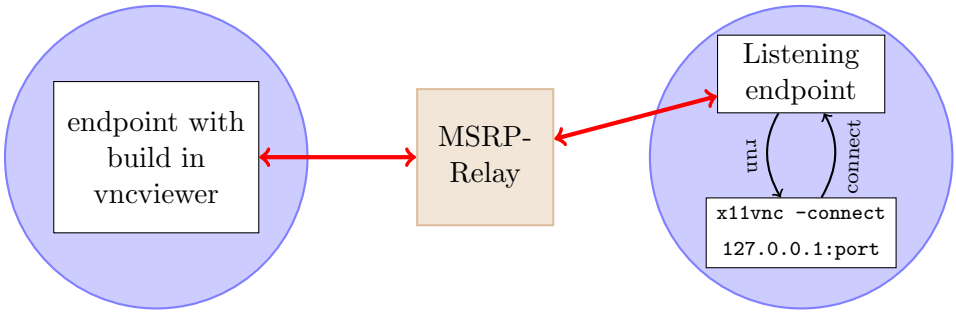


Figure 24: Final solution

In section 5.2 mentioned the "reverse connection" modus of operation of vncservers and vncviewers. It doesn't matter if the vncviewer or the vncserver initiated the connection, the vncserver is always the first to send a message. The "reverse connection" mode also has the advantage that it does not **require** the client to provide a password (although it could). SIP authorization is thus sufficient.

A setup using an connecting vncserver and vncclient (and thus listening endpoints), would be a good solution. In section 5.2 I also mentioned the pure python vncviewer. When using this vncviewer, we not only have no TCP sessions to manage, but also no running programs.

The most robust operation, from program and TCP session management perspective, is thus the one presented in figure 24.

The only disadvantage is the limited functionality and performance of the pure python vncviewer. A setup using a connecting vncserver and vncclient, might from a user perspective be even better.

6 Conclusion

SIP operates in a hostile environment. Its original goals were User location, User availability, User capabilities, Session setup and Session management. Firewalls and NATs were not commonplace when SIP was originally designed. The IETF has addressed these issues and have come up with a working solution for SIP.

However, SIP does establishing the actual media session that is initiated using SIP. The IETF has several proposals that can help SIP User Agents to address these issues, but they are not widely deployed yet.

In the mean time, industry has build working solutions for SIPs primarily usage at the moment: voice and video calls, using RTP. They have the disadvantage that they break SIPs principle of having no intelligence about the sessions it sets up at all. As a consequence they reduce SIPs extensibility.

From a standards perspective this is bad, but if it is also bad enough for SIP infrastructure deployers remains to be seen.

To answer the research question, *"How can application and desktop sharing, initiated by SIP, be realised in existing SIP infrastructure with the least possible impact on that infrastructure"*:

By using RTP as its transport protocol.

The draft-boyaci-avt-app-sharing-00 draft proposes a standard for application sharing over RTP. It allows the usage of UDP as the underlying transport for both multicast and unicast session, but it recommends using TCP for unicast session. The TCP based sessions using this protocol will probably not work in existing industry solutions.

If there will be enthusiasm to implement a brand new application sharing standard when there already exist well established publicly available standards such as the RFB protocol, also remains to be seen.

In this research project we have also explored the usage of MSRP to transport application sharing protocols (RFB specifically). MSRP addresses the NAT-Traversal problem with a well defined standard. Also, MSRP itself is *the* standard for instant messaging with SIP.

Instant messaging *is* very popular. It would mean substantial added value for SIP deployers to support it. I have not investigated how widely MSRP is supported, but if it is available, the solution explored in sections 4 and 5 (RFB over MSRP) is a solution that can be used and is working *now*.

Glossary

address-of-record In SIP, the address-of-record is the regular SIP address with which an end-user may contact another end-user. A SIP Proxy for the domain of the address-of-record should be able to find one or more contact-addresses for that address-of-record, to make the connection to the UA's of the end-user with that address-of-record, 5

Application Sharing The sharing of the graphical user-interface of an application amongst multiple users simultaneously in real time, 3

contact-address In SIP, a contact-address denotes a host, port, protocol and id on which a SIP end-user with an address-of-record can be contacted. A Registrar binds a SIP address-of-record to one or more contact-addresses, 5

Desktop Sharing The sharing of a computer desktop amongst multiple users simultaneously in real time, 3

EMSSC *Enhanced Mail System Status Codes*[10]: A standard for status codes consisting of three decimals. The first indicates the class of the status: `2` for success, `4` for request failure and `5` for server failure. The second digit indicates the subject of the status code and the third the details. SIP (and HTTP) use this format for the response codes in their transaction model. In SIP (and HTTP), the class digit is extended with: `1` for provisional responses (request in progress), `3` for redirection and `6` for global failures, 7

HTTP *Hypertext Transfer Protocol*[8]: The communication protocol for the World Wide Web. SIP messages look a lot like HTTP messages. They follow a request/response transaction model. Requests and responses are formated using the Internet Message Format, but are preceded by an extra line. In case of the request, this first line contains the request *method*, the method arguments, and ends with a protocol specifier. The first line of a response consists of a protocol specifier, followed by a status code and a textual representation of that status (much like the Enhanced Mail System Status Codes standard), 7

*These descriptions are shamelessly copied from Wikipedia on 30th January 2009

IANA *Internet Assigned Numbers Authority*: The entity that oversees global IP address allocation, root zone management for the Domain Name System (DNS), media types, and other Internet protocol assignments*, 3

IETF *Internet Engineering Task Force*: A large open international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet. See http://www.ietf.org/, 3

IM *Instant Messaging*: A form of real-time communication between two or more people based on typed text*, 4

Internet Message Format A message formated with this standard[9], consists of a set of headers, optionaly followed by a empty line and the message payload. HTTP and SIP use this format in their transaction model, 7, 23

ITU *International Telecommunication Union*: International organization established to standardize and regulate international radio and telecommunications*, 36

Location service In SIP a Location service is the database in which the address-of-record bindings to their contact-addresses are stored. Registrars and Proxies for a domain must have access to the same Location service, 8

MIME *Multipurpose Internet Mail Extensions*: An Internet standard that extends the format of e-mail to support:

- Text in character sets other than ASCII
- Non-text attachments
- Message bodies with multiple parts
- Header information in non-ASCII character sets

MIME's use, however, has grown beyond describing the content of e-mail to describing content type in general* See [26, 27], 3

NAPTR A *Name Authority Pointer* is a record type in DNS. It adds a extensible mechanism to DNS for providing different types of information without having to use a new DNS record type. In SIP, NAPTR records are used to specify the registrars for a certain domain, 6

NAT *Network Address Tanslation*: Is a technique with which the network address information is altered in a packet when it passes through a router. Most common usage is network Masquerading, which means that the (private) source addresses of a network are altered to a single (public) address. With this technique an internet connection which services a single IP-address (as is common with consumer ADSL and cable subscriptions), can have multiple computers use that internet connection, 13

Network Transparency The ability to have the user-interface of an remotely started application presented on a local display. The remote application is started on a remote server from a local computer. The user-interface of that application is then presented on the computer screen of the local computer. Network transparency is a feature found in the X-windows system and RDP. It is a subset of the features of application sharing, 3

PSTN *Public Switched Telephone Network*: The network of the world's public circuit-switched telephone networks*, 4

RDP *Remote Desktop Protocol*: A protocol to be used for desktop or application sharing, originally based on the ITU-T.128 protocol[28], 4

Redirect server In SIP, a Redirect server is the network element that redirects address-of-records to other URLs. Often those other URLs are again address-of-records, but they don't have to be. How an end-user configures a redirection for its address-of-record is not specified with SIP., 11

RFB *Remote Framebuffer*: a simple protocol for remote access to graphical user interfaces* See [2], 4

RTP The *Real-time Transport Protocol*[29] defines a standardized packet format for delivering audio and video over the Internet*, 11

SDP *Session Description Protocol*[11]: SDP is used to agree on the type of media used and on the ip-addresses, port numbers and protocols used. SDP uses a request/response transaction model. First a party is contacted using a signalling protocol, such as SIP, to request a session for a certain media type. If the contacted party agrees on the media type, it returns (again via the signalling protocol), a ip-address, port number and protocol to be used to setup the session, 8

SIP *Session Initiation Protocol*, 3

SIP Proxy In SIP, a Proxy is the network element which can be contacted for a address-of-record. It forwards the requests to that address-of-records contact-address, 8

SIP Registrar In SIP, a Registrar is the network element with which the UA registers and authenticates its address-of-record. The Registrar binds that address then to the UA's, contact-address (its host location), 6

UA User Agent: The software or device setting up a SIP session on behalf of the end-user, 4

VOIP *Voice over Internet Protocol*: General term for a family of transmission technologies for delivery of voice communications over IP networks such as the Internet*, 4

References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393.

[2] Tristan Richardson. The RFB Protocol. http://www.realvnc.com/docs/rfbproto.pdf, August 2008.

[3] J. Rosenberg and H. Schulzrinne. Session Initiation Protocol (SIP): Locating SIP Servers. RFC 3263 (Proposed Standard), June 2002.

[4] M. Mealling. Dynamic Delegation Discovery System (DDDS) Part Three: The Domain Name System (DNS) Database. RFC 3403 (Proposed Standard), October 2002.

[5] A. Gulbrandsen, P. Vixie, and L. Esibov. A DNS RR for specifying the location of services (DNS SRV). RFC 2782 (Proposed Standard), February 2000.

[6] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[7] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999.

[8] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFC 2817.

[9] P. Resnick. Internet Message Format. RFC 2822 (Proposed Standard), April 2001. Obsoleted by RFC 5322, updated by RFCs 5335, 5336.

[10] G. Vaudreuil. Enhanced Mail System Status Codes. RFC 1893 (Proposed Standard), January 1996. Obsoleted by RFC 3463.

[11] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566 (Proposed Standard), July 2006.

[12] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. RFC 1918 (Best Current Practice), February 1996.

[13] J. Rosenberg and H. Schulzrinne. An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing. RFC 3581 (Proposed Standard), August 2003.

[14] C. Jennings and R. Mahy. Managing Client Initiated Connections in the Session Initiation Protocol (SIP). http://www.ietf.org/internet-drafts/draft-ietf-sip-outbound-16.txt, October 2008.

[15] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard), October 2008.

[16] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard), March 2003. Obsoleted by RFC 5389.

[17] R. Mahy J. Rosenberg and P. Matthews. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). http://www.ietf.org/internet-drafts/draft-ietf-behave-turn-12.txt, November 2008.

[18] C. Jennings, R. Mahy, and A. B. Roach. Relay Extensions for the Message Sessions Relay Protocol (MSRP). RFC 4976 (Proposed Standard), September 2007.

[19] B. Campbell, R. Mahy, and C. Jennings. The Message Session Relay Protocol (MSRP). RFC 4975 (Proposed Standard), September 2007.

[20] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle. Session Initiation Protocol (SIP) Extension for Instant Messaging. RFC 3428 (Proposed Standard), December 2002.

[21] G. Klyne and D. Atkins. Common Presence and Instant Messaging (CPIM): Message Format. RFC 3862 (Proposed Standard), August 2004.

[22] M. Garcia-Martin, M. Isomaki, G. Camarillo, S. Loreto, and P. Kyzivat. A Session Description Protocol (SDP) Offer/Answer Mechanism to Enable File Transfer. http://www.ietf.org/internet-drafts/draft-ietf-mmusic-file-transfer-mech-10.txt, January 2009.

[23] O. Boyaci and H. Schulzrinne. RTP Payload format for Application and Desktop Sharing. http://www.ietf.org/internet-drafts/draft-boyaci-avt-app-sharing-00.txt, October 2008.

[24] M. Garcia-Martin and J. Ott. Session Description Protocol (SDP) Extensions and Conventions for Collaboration Applications. http://www.netlab.tkk.fi/~jo/papers/draft-garcia-mmusic-sdp-collaboration-00.txt, February 2008.

[25] D. Yon and G. Camarillo. TCP-Based Media Transport in the Session Description Protocol (SDP). RFC 4145 (Proposed Standard), September 2005. Updated by RFC 4572.

[26] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies. RFC 2045 (Draft Standard), November 1996. Updated by RFCs 2184, 2231, 5335.

[27] N. Freed and N. Borenstein. Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types. RFC 2046 (Draft Standard), November 1996. Updated by RFCs 2646, 3798, 5147.

[28] T.128. http://www.itu.int/rec/T-REC-T.128/en, February 1998.

[29] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003.