Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Multimedia Session Establishment Protocols

draft-ietf-mmusic-ice-02

Status of this Memo

By submitting this Internet-Draft, I certify that any applicable patent or other IPR claims of which I am aware have been disclosed, and any of which I become aware will be disclosed, in accordance with RFC 3668.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on January 17, 2005.

Copyright Notice

Abstract

This document describes a methodology for Network Address Translator (NAT) traversal for multimedia session signaling protocols, such as the Session Initiation Protocol (SIP). This methodology is called Interactive Connectivity Establishment (ICE). ICE makes use of existing protocols, such as Simple Traversal of UDP Through NAT (STUN) and Traversal Using Relay NAT (TURN). ICE makes use of STUN in peer-to-peer cooperative fashion, allowing participants to discover, create and verify mutual connectivity.

Table of Contents

1. Introduction

A multimedia session signaling protocol is a protocol that exchanges control messages between a pair of agents for the purposes of establishing the flow of media traffic between them. This media flow is distinct from the flow of control messages, and may take a different path through the network. Examples of such protocols are the Session Initiation Protocol (SIP) [3], the Real Time Streaming Protocol (RTSP) [5] and the International Telecommunications Union (ITU) H.323.

These protocols, by nature of their design, are difficult to operate through Network Address Translators (NAT). Because their purpose in life is to establish a flow of packets, they tend to carry IP addresses within their messages, which is known to be problematic through NAT [6]. The protocols also seek to create a media flow directly between participants, so that there is no application layer intermediary between them. This is done to reduce media latency, decrease packet loss, and reduce the operational costs of deploying the application. However, this is difficult to accomplish through NAT. A full treatment of the reasons for this is beyond the scope of this specification.

Numerous solutions have been proposed for allowing these protocols to operate through NAT. These include Application Layer Gateways (ALGs), the Middlebox Control Protocol [7], Simple Traversal of UDP through NAT (STUN) [1], Traversal Using Relay NAT [16], Realm Specific IP [8][9], symmetric RTP [10], along with session description extensions needed to make them work, such as [2]. Unfortunately, these techniques all have pros and cons which make each one optimal in some network topologies, but a poor choice in others. The result is that administrators and implementors are making assumptions about the topologies of the networks in which their solutions will be deployed. This introduces a lot of complexity and brittleness into the system. What is needed is a single solution which is flexible enough to work well in all situations.

This specification provides that solution. It is called Interactive Connectivity Establishment, or ICE. ICE makes use of many of the protocols above, but uses them in a specific methodology which avoids many of the pitfalls of using any one alone. ICE uses STUN and TURN without extension, and allows for other similar protocols to be used as well. However, it does require additional signaling capabilities to be introduced into the multimedia session signaling protocols. For those protocols which make use of the Session Description Protocol (SDP), this specification defines the necessary extensions to it. Other protocols will need to define their own mechanisms.

2. Multimedia Signaling Protocol Abstraction

This specification defines a general methodology that allows the media streams of multimedia signaling protocols to successfully traverse NAT. This methodology is independent of any particular signaling protocol. In order to discuss the methodology, we need to to define an abstraction of a multimedia signaling system, and define terms that can be used throughout this specification. Figure 1 shows the abstraction.

```
                               +-----------+
                               |           |
                               |           |
                             > | Signaling |\
                            /  | Relay     | \
                           /   |           |  \
                Initiate  /    |           |   \   Initiate
               Message   /   / +-----------+    \  Message
                        /   /                <   \
                       /   /                  \   \
                      /   /                    \   \
                     /   / Accept       Accept  \   \
                    /   /   Message     Message  \   >
                   /   /                          \
       +-----------+  /                            \   +-----------+
       |           | <                                 |           |
       |           |             Media Stream          |           |
       |  Session  | ................................  | Session   |
       | Initiator |                                   | Responder |
       |           |             Media Stream          |           |
       |           | ................................  |           |
       +-----------+                                   +-----------+
```

Figure 1

Communications occur between two clients - the session initiator and the session responder, also referred to as the initiator and responder. The initiator is the one that decides to engage in communications. To do so, it sends an initiate message. The initiate message contains parameters that describe the capabilities and configuration of media streams for the initiator. This message may travel through signaling intermediaries, called a signaling relay, before finally arriving at the session responder. Assuming the session responder wishes to communicate, it generates an accept message, which is relayed back to the initiator. This message contains capabilities and configuration of media streams for the responder. As a result, media streams are established between the initiator and responder. The signaling protocol may also support an operation that allows for termination of the communications session. We refer to this signaling message as a terminate message.

This abstraction is readily mapped to SIP, RTSP, and H.323, amongst others. For SIP, the initiator is the User Agent Client (UAC), the responder is the User Agent Server (UAS), the initiate message is a SIP message containing an SDP offer (for example, an INVITE), the accept message is a SIP message containing an SDP answer (for example, a 200 OK), and the terminate message is a BYE. For RTSP, the initiator is the RTSP client, the responder is the RTSP server, the initiate message is a SETUP message, and the accept message is a SETUP response.

This specification defines parameters that need to be included in these various signaling messages in order to implement the functionality described by ICE. Those parameters are represented in XML for convenience. Any multimedia signaling protocol that uses ICE will need to define how to map those parameters into its own protocol messages. Section 9 provides such a mapping for SIP.

3. Terminology

Several new terms are introduced in this specification:

Session Initiator: A software or hardware entity that, at the request of a user, tries to establish communications with another entity, called the session responder. A session initiator is also called an initiator.

Initiator: Another term for a session initiator.

Session Responder: A software or hardware entity that receives a request for establishment of communications from the session initiator, and either accepts or declines the request. A session responder is also called a responder.

Responder: Another term for a session responder.

Client: Either the initiator or responder.

Peer: From the perspective of one of the clients in a session, its peer is the other client. Specifically, from the perspective of the initiator, the peer is the responder. From the perspective of the responder, the peer is the initiator.

Signaling Relay: An intermediary of signaling messages. Examples are SIP proxies and H.323 Gatekeepers.

Initiate Message: The signaling message used by an initiator to establish communications. It contains capabilities and other information needed by the responder to send media to the initiator.

Accept Message: The signaling message used by a responder to agree to communications. It contains capabilities and other information needed by the initiator to send media to the responder.

Terminate Message The signaling message used by a client to terminate the session and associated media streams.

Transport Address: The combination of an IP address and port.

Local Transport Address: A local transport address is a transport address that has been allocated from the operating system on the host. This includes transport addresses obtained through Virtual Private Networks (VPNs) and transport addresses obtained through Realm Specific IP (RSIP) [8] (which lives at the operating system level). Transport addresses are typically obtained by binding to an interface.

Derived Transport Address: A derived transport address is a transport address which is associated with, but different from, a local transport address. The derived transport address is associated with the local transport address in that packets sent to the derived transport address are received on the socket bound to that local transport address. Derived addresses are obtained using protocols like STUN and TURN, and more generally, any UNSAF protocol [11].

Peer Derived Transport Address: A peer derived transport address is a derived transport address learned from a STUN server running within a peer in a media session.

TURN Derived Transport Address: A derived transport address obtained from a TURN server.

STUN Derived Transport Address: A derived transport address obtained from a STUN server whose address has been provisioned into the UA. This, by definition, excludes Peer Derived Transport Addresses.

Unilateral Allocations: Queries made to a network server which provides an UNSAF service.

Bilateral Allocations: Addresses obtained by using an UNSAF service that actually runs on the peer of the communications session. Peer derived transport addresses are synonymous with bilateral allocations.

4. Overview of ICE

ICE makes the fundamental assumption that clients exist in a network of segmented connectivity. This segmentation is the result of a number of addressing realms in which a client can simultaneously be connected. We use "realms" here in the broadest sense. A realm is defined purely by connectivity. Two clients are in the same realm if, when they exchange the addresses each has in that realm, they are able to send packets to each other. This includes IPv6 and IPv4 realms, which actually use different address spaces, in addition to private networks connected to the public Internet through NAT.

The key assumption in ICE is that a client cannot know, apriori, which address realms it shares with any peer it may wish to communicate with. Therefore, in order to communicate, it has to try connecting to addresses in all of the realms.

Before the initiator establishes a session, it obtains as many IP address and port combinations in as many address realms as it can. These adresses all represent potential points at which the initiator will receive a specific media stream. Any protocol that provides a client with an IP address and port on which it can receive traffic can be used. These include STUN, TURN, RSIP, and even a VPN. The client also uses any local interface addresses. A dual-stack v4/v6 client will obtain both a v6 and a v4 address/port. The only requirement is that, across all of these addresses, the initiator can be certain that at least one of them will work for any responder it might communicate with. Unfortunately, if the initiator communicates with a peer that doesn't support ICE, only one address can be provided to that peer. As such, the client will need to choose one default address, which will be used by non-ICE clients. This would typically be a TURN derived transport address, as it is most likely to work with unknown non-ICE peers.

The initiator then runs a STUN server on each of the local transport addresses it has obtained. The initiator will need to be able to demultiplex STUN messages and media messages received on that IP address and port, and process them appropriately. All of these addresses are placed into the initiate message, and they are ordered in terms of preference. Preference is a matter of local policy, but typically, lowest preference would be given to transport addresses learned from a TURN server (i.e., TURN derived transport addresses). The initiate message also conveys the STUN username and password which are required to gain access to the STUN server on each address/

port combination.

The initiate message is sent to the responder. This specification does not address the issue of how the signaling messages themselves traverse NAT. It is assumed that signaling protocol specific mechanisms are used for that purpose. The responder follows a similar process as the initiator followed; it obtains addresses from local interfaces, STUN servers, TURN servers, etc., and it places all of them into the accept message.

Once the responder receives the initiate message, it has a set of potential addresses it can use to communicate with the initiator. The initiator will be running a STUN server at each address. The responder sends a STUN request to each address, in parallel. When the initiator receives these, it sends a STUN response. If the responder receives the STUN response, it knows that it can reach its peer at that address. It can then begin to send media to that address. As additional STUN responses arrive, the responder will learn about additional transport addresses which work. If one of those has a higher priority than the one currently in use, it starts sending media to that one instead. No additional control messages (i.e., SIP signaling) occur for this change.

The STUN messages described above happen while the accept message is being sent to the intitiator. Once the intitiator receives the accept message, it too will have a set of potential addresses with which it can communicate to the responder. It follows exactly the same process described above.

Furthermore, when a either the initiator or responder receives a STUN request, it takes note of the source IP address and port of that request. It compares that transport address to the existing set of potential addresses. If it's not amongst them, it gets added as another potential address. The incoming STUN message provides the client with enough context to associate that transport address with a STUN username, STUN password, and priority, just as if it had been sent in an initiate or accept message. As such, the client begins sending STUN messages to it as well, and if those succeed, the address can be used if it has a higher priority.

After a successful STUN transaction, the client will re-perform the STUN query periodically to revalidate connectivity. This allows for recovery from NAT failures, or from route flaps which may cause packets to suddenly traverse a different NAT. As such, the address used as the destination for media is the highest priority address to which connectivity currently exists.

5. Detailed ICE Algorithm

This section describes the detailed processing needed for ICE.

5.1 Initiator Processing

5.1.1 Sending the Initiate Message

When the initiator wishes to begin communications, it starts by gathering transport addresses, as described in Section 5.3.1, and starting a STUN server on each local transport address, as described in Section 5.3.2. This process can actually happen at any time before sending an initiate message. A client can pre-gather transport addresses, using a user interface cue (such as picking up the phone, or entry into an address book) as a hint that communications is imminent.

When it comes time to initiate communications, it determines a priority for each one and identifies one as a default, as described in Section 5.3.3.

The next step is to construct the initiate message. Section 7 provides the XML schema for the initiate message. The message consists of a series of media streams. For each media stream, there is a default address and a list of alternates. The default address is the one that will be used by responders that don't understand ICE (for SIP, this is accomplished by mapping the default address into the m and c line in the SDP). The alternates represent addresses that the responder should also try. In SIP, these are conveyed with the new SDP alt parameter.

The client then encodes all of its available transport addresses (including the default) as a series of alternate elements. Each alternate element conveys a transport address for RTP, one for RTCP, a STUN username fragment and STUN password. The client MUST assign each alternate a unique identifier. These identifiers MUST be unique across all alternates used within the session. This identifier is encoded in the "id" attribute of the alternate element. The priority for the transport address, as computed above, is included as an attribute as well.

Once the initiate message is constructed, it is sent.

5.1.2 Processing the Accept

There are two possible cases for processing of the Accept message. If the recipient of the Initiate message did not support ICE, the Accept message will only contain the default address information. As a result, the initiator knows that it cannot perform its connectivity checks. In this case, it SHOULD just send to the transport address listed. However, if local configuration information tells the initiator to try connectivity checks by sending them through the TURN server, this means that packets sent directly to responder may be dropped by a local firewall. To deal with this, the initiator SHOULD issue a SEND command using this new transport address. The SEND command contains the media packet to send to the responder. Once this command has been accepted, the initiator SHOULD send all media packets to the TURN server, which will then forward them towards the responder.

If the Accept message contains alternates, it implies that the responder supported ICE. In that case, the initiator takes each transport address, STUN username, STUN password and priority, and places them into a list, called the candidate list. It then begins processing the candidate list as described in Section 5.3.4. That processing associates a state with each transport address. As described there, once a successful STUN query is made to the STUN server at an address, the initiator can begin sending media to that address.

5.2 Responder Processing

5.2.1 Processing the Initiate Message

Upon receipt of the initiate message, the client starts gathering transport addresses, as described in Section 5.3.1, and starts a STUN server on each local transport address, as described in Section 5.3.2. This processing is done immediately on receipt of the request, to prepare for the case where the user should accept the call, or early media needs to be generated.

At some point, the responder will decide to accept or reject the communications. A rejection terminates ICE processing, of course. In the case of acceptance, the accept message is constructed as follows.

The client first determines a priority for each transport address it has gathered, and identifies one as a default, as described in Section 5.3.3.

Constructing the accept proceeds identically to the way in which the initiate message is constructed (Section 5.1.1).

The accept is then sent.

5.3 Common Procedures

This section discusses procedures that are common between initiator and responder.

5.3.1 Gathering Transport Addresses

A client gathers addresses when it believes that communications is imminent. For initiators, this occurs before sending an initiate message (Section 5.1.1). For responders, it occurs before sending a accept message (Section 5.2.1).

There are two types of addresses a client can gather - local transport addresses, and derived transport addresses. Local transport addresses are obtained by binding to an ephemeral port on an interface (physical or virtual) on the host. A multi-homed host SHOULD attempt to bind on all interfaces for all media streams it wishes to receive. For media streams carried using the Real Time Transport Protocol (RTP) [12], the client will need to bind to an ephemeral port for both RTP and RTCP.

The result will be a set of local transport addresses. The client may also have access to servers that provide unilateral self-address fixing (UNSAF) [11]. Examples of such protocols include STUN, TURN, and TEREDO [15]. All ICE implementations MUST implement STUN and TURN, but MAY, through configuration, disable the use of STUN or TURN for unilateral address allocation (STUN is mandatory for the connectivity checks described below). When disabled, it MUST be possible through user or administrator operation to re-enable. This allows all implementations to have the breadth of protocol support needed to work in all situations, with the flexibility to turn if off if its not needed.

These protocols work by having the client send, from a specific local transport address, some kind of message to a server. The server provides to the client, in some kind of response, an additional transport address, called a derived transport address. This derived transport address is derived from the local transport address. Here, derivation means that a request sent to the derived transport address might (under good network conditions) reach the client on its local transport address.

For each of these protocols, the client may have access to a multiplicity of servers. For example, a user connected to a natted cable access network might have access to a STUN server in the private cable network and in the public Internet. For each local transport address, the client SHOULD obtain an address from every server for each protocol it supports. The result of this will be a set of derived transport addresses, with each derived address associated with the local transport address it is derived from.

5.3.2 Enabling STUN on Each Local Transport Address

Once the client has obtained a set of transport addresses, it starts a STUN server on each local transport address (including ones used for RTCP). This, by definition, means that the STUN service will be reached for requests sent to the derived addresses.

However, the client does not need to provide STUN service on any other IP address or port, unlike the STUN usage described in [1]. The need to run the service on multiple ports is to support the change flags. However, those flags are not needed with ICE, and the server SHOULD reject, with a 400 response, any STUN requests with these flags set.

Furthermore, there is no need to support TLS or to be prepared to receive SharedSecret request messages. Those messages are used to obtain shared secrets to be used with BindingRequests. However, with ICE, usernames and passwords are exchanged in the signaling protocol.

The client will receive both STUN requests and media packets on each local transport address. The client MUST be able to disambiguate them. In the case of RTP/RTCP, this disambiguation is easy. RTP and RTCP packets start with the bits 0b10 (v=2). The first two bits in STUN are always 0b00. This disambiguation also works for packets sent using Secure RTP [13], since the RTP header is in the clear. Disambiguating STUN with other media stream protocols may be more complicated. However, it can always be possible with arbitrarily high probabilities by selecting an appropriately random username (see below).

The need to run STUN on the same transport address as the media stream represents the "ugliest" piece of ICE. However, it is an essential part of the story. By sending STUN requests to the very same place media is sent, any bindings learned through STUN will be useful even when communicating through symmetric NATs. This results in a substantial increase in the scope of applicability of STUN.

For each local transport address where a STUN server is running, the client MUST choose a username fragment and a password. The username fragment created by the client will be concatenated with the fragment created by its peer. The result will serve as the username provided by its peer in STUN requests. By creating the username as a combination of information from each side of a call, it allows a client to correlate the source of the request with a candidate transport address. This is discussed further below.

The username fragment MUST be globally unique, so that no other host will select a username with the same value. This username fragment and password will be passed to its peer in an initiate or accept message. As such, the process described in this section will associate, with each local transport address, a username fragment and password. The client also associates this same username fragment and password with any transport addresses derived from the local transport address.

The global uniqueness requirement stems from the lack of uniquenes afforded by IP addresses. Consider clients A, B, and C. A and B are within private enterprise 1, which is using 10.0.0.0/8. C is within private enterprise 2, which is also using 10.0.0.0/8. As it turns out, B and C both have IP address 10.0.1.1. A initiates communications to C. C, in its accept message, provides A with its transport addresses. In this case, thats 10.0.1.1:8866 and 8877. As it turns out, B is in a session at that same time, and is also using 10.0.1.1:8866 and 8877. This means that B has a STUN server running on those ports, just as C does. A will send a STUN request to 10.0.1.1:8866 and 8877. However, these do not go to C as expected. Instead, they go to B. If B just replied to them, A would believe it has connectivity to C, when in fact it has connectivity to a completely different user, B. To fix this, the STUN username takes on the role of a unique identifier. C provides A with a unique username. A uses this username in its STUN query to 10.0.1.1:8866. This STUN query arrives at B. However, the username is unknown to B, and so the request is rejected. A treats the rejected STUN request as if there were no connectivity to C (which is actually true). Therefore, the error is avoided.

Once the STUN server is started, it MUST run continuously until the session is completed. While the server is running, it MUST act as a normal STUN server, but MUST only accept STUN requests from clients that authenticate, as discussed below in Section 5.3.5

5.3.3 Prioritizing the Transport Addresses and Choosing a Default

The prioritization process takes a list of transport addresses, and associates each with a priority. This priority reflects the desire that the UA has to receive media on that address, and is assigned as a value from 0 to 1 (1 being most preferred). Priorities are ordinal, so that their significance is only relative to other transport address priorities in the same list.

This specification makes no normative recommendations on how the prioritization is done. However, some useful guidelines are suggested on how such a prioritization can be determined.

One criteria for choosing one transport address over another is whether or not that transport address involves the use of a relay. That is, if media is sent to that transport address, will the media first transit a relay before being received. TURN derived transport addresses make use of relays (the TURN server), as to any local transport addresses associated with a VPN server. When media is transited through a relay, it can increase the latency between transmission and reception. It can increase the packet losses, because of the additional router hops that may be taken. It may increase the cost of providing service, since media will be routed in and right back out of a relay run by the provider. If these concerns are important, transport addresses with this property can be listed with lower priority.

Another criteria for choosing one address over another is IP address family. ICE works with both IPv4 and IPv6. It therefore provides a transition mechanism that allows dual-stack hosts to prefer connectivity over IPv6, but to fall back to IPv4 in case the v6 networks are disconnected (due, for example, to a failure in a 6to4 relay) [14]. It can also help with hosts that have both a native IPv6 address and a 6to4 address. In such a case, higher priority could be afforded to the native v6 address, followed by the 6to4 address, followed by a native v4 address. This allows a site to obtain and begin using native v6 addresss immediately, yet still fallback to 6to4 addresses when communicating with clients in other sites that do not yet have native v6 connectivity.

Another criteria for choosing one address over another is security. If a user is a telecommuter, and therefore connected to their corporate network and a local home network, they may prefer their voice traffic to be routed over the VPN in order to keep it on the local network when communicating within the enterprise, but use the local network when communicating with users outside of the enterprise.

Another criteria for choosing one address over another is topological awareness. This is most useful for transport addresses which make use of relays (including TURN and VPN). In those cases, if a client has preconfigured or dynamically discovered knowledge of the topological proximity of the relays to itself, it can use that to select closer relays with higher priority.

Once the transport addresses have been prioritized, one is selected as the default. This is the address that will be used by a peer that doesn't understand ICE. The default has no relevance when communicating with an ICE capable peer. As such, it is RECOMMENDED that the default be chosen based on the likelihood of that address being useful when communicating with a peer that doesn't support ICE.

This will frequently be a TURN derived transport address from a TURN server providing public IP addresses.

5.3.4 Sending STUN Connectivity Checks

Once a responder has received an initiate message, or an initiator has received an accept message, the list of transport addresses is extracted from the message. These transport addresses, called the remote transport addresses, along with the username fragment, password, and priority from the message are placed into a table, called the candidate table. There is a candidate table for RTP for each media stream, and for RTCP for each media stream. So, if a session is established with audio and video, there would be four tables - audio RTP, audio RTCP, video RTP and video RTCP.

The client then takes its own gathered addresses, and creates a subset called the sourceable addresses. This subset is the set of local transport addresses (including VPN and RSIP) and TURN derived transport addresses. Thus, it excludes STUN derived transport addresses. The formal definition of this subset is defined below.

Each row in this table is then replicated once for each sourceable transport address. The table has a column for the sourceable transport address value, and this is populated upon replication. That table also has a column called "my username fragment", which is the username fragment that the client created for sourceable transport address in that row. Each row in this table is called a candidate.

Each candidate is associated with a state. The state represents the current understanding of connectivity to that remote transport address when packets are sent from that sourceable address. There are five possible states. These states are:

INIT: No STUN transaction has been completed towards this remote transport address from this sourceable address.

HANDSHAKING: One or more STUN transactions have failed, but insufficient time has passed since leaving the INIT state to be certain that the remote transport address is unreachable from this sourceable address. This state is important for connectivity checks made to STUN derived transport addresses through port restricted NAT or a TURN derived transport address.

BAD: All STUN transactions to this remote transport address from this sourceable address have either timed out, or failed with a 600 response, and a sufficient amount of time has elapsed since the INIT state to have high confidence that the remote transport address cannot be reached from this sourceable address.

GOOD: The last STUN transaction to this remote transport address from this sourceable address was successful. However, it is not the highest priority candidate, and therefore, is not in use for media.

When the client first populates the tables from the initiate or accept message, all of the transport addresses are set to the INIT state.

Consider the the following example. An initiator sends an initiate message with one media stream (audio), with two transport addresses, A and B. A is a local transport address, and B is a STUN derived transport address (although that fact is not signaled in the message). Both of these will have the same username fragment and password, but different priorities. The initiate message is sent to the responder. The responder has a local transport address, a STUN derived transport address, and a TURN derived transport address. Call these X, Y and Z respectively. Thus, it has two sourceable addresses, X and Z. The table created by the responder would have four rows. Each of the two transport addresses in the initiate message is present twice, once with the responder's local transport address, and once with its TURN derived address. Such a table might look like this:

Remote	Srcable	User Frag	Passwd	My-Usr-Frag	Priority	State
A	X	asd9f8f8==	siprulz	x-frag	0.4	INIT
A	Z	asd9f8f8==	siprulz	z-frag	0.4	INIT
B	X	asd9f8f8==	siprulz	x-frag	0.2	INIT
B	Z	asd9f8f8==	siprulz	z-frag	0.2	INIT

The client begins a STUN BindingRequest transaction for each candidate. This STUN transaction is sent to the IP address and port from the Remote column. It sends the request from the IP address and port in the sourceable column. For local transport addresses, that means sending from the locally bound socket. For VPN addresses, that means sending from the socket bound to the VPN interface. For TURN derived transport addresses, this means using the TURN Send message to send a request through the TURN server. This provides the definition of the sourceable flag: they represent distinct transport addresses that a client can send from. A STUN derived transport address is not distinct from a local transport address, since a client cannot send a packet to a particular IP address and port with different source IP addresses and ports as seen by that recipient [[REPHRASE]]

The STUN USERNAME attribute MUST be present. It is set to the concatenation of the user fragment from the table, with the "My User Fragment" from the candidate. Thus, for the candidate with remote transport address A and sourceable address X, the USERNAME would be set to "asd9f8f8==x-frag". The BindingRequest SHOULD contain a MESSAGE-INTEGRITY attribute, computed using the username in the USERNAME attribute, and the password from the password field in the row. The BindingRequest MUST NOT contain the CHANGE-REQUEST or RESPONSE-ADDRESS attribute.

Each of these STUN transactions will generate either a timeout, or a response. If the response is an error, but recoverable as described in RFC 3489, the client SHOULD try again using the procedures discussed there. Either initialy, or after retry, the STUN transaction will produce a timeout result, a success result, or a non-recoverable failure result (error codes 400, 431, or 600). These correspond to "timeout", "success", and "error" events, respectively.

These events are fed into the state machine described in Figure 3. This figure shows the transitions between states that occur on the completion of the STUN BindingRequest transaction. After the completion of each transaction, the client sets a timer that determines when it will do another transaction for that candidate. The result of that next transaction drives the next transition in the state machine, and so on. Since timers are set at the entry to each state, STUN BindingRequest tranasactions will be tried continuously throughout a call. This is necessary to detect a variety of failure cases, as discussed below.

```
                                                   ..........
                                                   .        . timeout/
                                                   .        .  Set Rapid
           +---------+                       +---------+    .  Retry Timer
           |         |                       |         |    .
           |         |                       |         |<....
           |  INIT   |......................>|  HAND   |
           |         | timeout/              | SHAKING |
           |         |  Set Rapid            |         |
           +---------+  Retry Timer,  error/ +---------+
             .  .       Giveup Timer   Set     .    .
             .  .                     Retry    .    .
      error/ .  .                     Timer    .    .
       Set   .  .  .............................    . success/
      Retry  .  .  .                                .  Set Refresh
      Timer  .  ...C..............................  .  Timer
             .     .           success/          .  .
             .     .            Set Refresh      .  .
             V     V            Timer            V  V
           +---------+                       +---------+
           |         |                       |         |
           |         |                       |         |
           |  BAD    |......................>|  GOOD   |
       ...>|         |        success/       |         |.......
       .   |         |         Set Refresh   |         |      .
       .   +---------+         Timer         +---------+      .
       .     .   ^                              .    ^        .
       .     .   .                              .    .        .
       .......   .                              .    ..........
     timeout or  ................................        success/
     error/                   timeout or                  Set Refresh
      Set                     error/                      Timer
     Retry                     Set
     Timer                    Retry
                              Timer
```

Figure 3

Starting in the INIT state, if the transaction is successful, the client has verified connectivity to that remote transport address when sending from that sourceable transport address. This means that media packets sent in exactly the same way will get through. As such, the FSM transitions to the GOOD state, and the client sets the Refresh Timer. This timer is used to continually check that a good candidate remains good. It is possible for a candidate to cease being good if a NAT should fail and recover, resulting in loss of any bindings it holds, or if an IP route should flap, causing those packets to be delivered through a new NAT that allocates new bindings, or a firewall with different policies. The Retry Timer value SHOULD be configurable. In order to rapidly recover from failures, it is RECOMMENDED that it default to five seconds. [[TODO: Need to work this number as a function of codec rates as well, perhaps apply the RTCP algorithm for its computation.]]

If, from the INIT state, the STUN transaction times out, the FSM enters the HANDSHAKE state. At this point, there are two reasons that the STUN request might have timed out. One reason is that the candidate is simply unreachable. The other reason is that the peer is behind a port restricted NAT, and so STUN requests from the client cannot get through until its peer creates a permission by generating its own STUN request. It may take some time to generate that STUN request, as it may depend on a response message getting delivered. As such, the HANDSHAKE state allows for rapid retry of the STUN transaction until enough time has passed to be certain that the remote transport address is actually unreachable. Thus, upon entering the HANDSHAKE state, two timers are set. The first, called the Rapid Retry timer, determines how long until the next attempt. This timer SHOULD be configurable. It is RECOMMENDED that it default to 1 second. The second timer, called the Giveup Timer, determines how long the client will keep trying until it decides that the remote transport address is unreachable. This timer SHOULD be configurable. It is RECOMMENDED that it default to 50 seconds. This is a reasonable approximation of the maximum SIP transaction duration.

If, from the INIT state, the STUN transaction generates an error, the FSM moves into the BAD state. The retry timer is set. This retry timer is used to periodically retry, and see if the candidate may now be reachable. The value of this timer SHOULD be configurable. It is RECOMMENDED that it default to 1 minute.

If, while in the HANDSHAKE state, the Giveup timer fires, or the STUN transaction results in an error, the client moves into the BAD state, and sets the retry timer. The default durations for ths timer are identical for all entries into the BAD state, and thus it defaults to 1 minute here as well. If, while in the HANDSHAKE state, the Rapid Retry timer fires, the timer is reset and the client remains in the HANDSHAKE state.

If, while in the BAD state, the retried transaction is executed and fails or results in a timeout, the client resets the timer and remains in the BAD state. If the STUN transaction succeeds, it moves into the GOOD state and sets the refresh timer. The default durations for this timer are the same for all entries into the GOOD state, and thus it defaults to 1 second.

If while in the GOOD state, the transaction resulting from the refresh timer times out or fails, the client moves into the BAD state and sets the retry timer. If, however, that transaction succeeds, the client stays in the GOOD state and resets the refresh timer.

As the FSM operates throughout the call, candidates will move their states around. At any point in time, the client sends media packets (including RTCP) using one of the candidates in the GOOD state. It is RECOMMENDED that the one with highest priority be used. It another candidate should change state such that it moves into the GOOD state, and it has a higher priority, the client SHOULD switch to that candidate, but SHOULD do so after waiting a small period of time (10 seconds is RECOMMENDED) to prevent against flapping of candidates during periods of route flaps in the network.

To send media to a candidate, the client sends media packets (whether they are RTP or RTCP or something else) to the remote transport address, from the sourceable transport address.

If, for some reason, there was at least one candidate in the GOOD state, and due to an FSM transition, none of the candidates are in the GOOD state, the client SHOULD forcefully transition all of the candidates into the HANDSHAKE state in an attempt to rapidly reconnect. If none of them succeed, and all of the candidates enter the BAD state, the client SHOULD terminate the call and alert the user to the failure [[TODO: Need to work in some good congestion control here; in cases where timeouts happen due to network congestion this is probably too agressive]].

5.3.5 Receiving STUN Requests

When a client receives a STUN request (presumably after disambiguating it from a media packet), it follows the logic described in this section.

The client MUST follow the procedures defined in RFC 3489 and verify that the USERNAME attribute is known to the server. Here, this is done by taking the USERNAME attribute, and doing a prefix match against the "my user fragment" column in the candidate table. If it doesn't match any rows, the client generates a 432 response. If it matches multiple rows, the client checks the suffix of the username against the "user fragment" column. If it doesn't match any rows, the client generates a 432 response. If it does match rows, it will match those rows corresponding to the transport addresses that the peer could have sent this STUN request from.

Assuming the USERNAME is valid, the client MUST generate a STUN response per RFC 3489.

Once the response is sent, the client examines the source IP and port where the request came from. It matches those against the remote transport addresses in the candidate table. If there is no match, this source address is itself another possible candidate. As with other candidates, it must be associated with a STUN username fragment, password and priority, all normally provided by the peer, along with sourceable transport addresses and their username fragments.

How does the client obtain this other information? The suffix of the USERNAME is the key (literally). That suffix was already provided to the client in an initiate or accept message, and was used to populate the current candidate table. If it matches an existing value in the table, it means that the STUN request came from the same transport address as a previously advertised candidate; however, when it showed up at the client, its source IP address was different than the peer thought it would be. This will happen when a symmetric NAT exists between the clients. In this case, the source IP address and port of the STUN packet now become a viable candidate, since the client should be able to send messages back to it and reach its peer.

However, this connectivity, like all other connectivity, needs to be verified. So, the client needs to find out the user fragment and password to use in STUN requests. To do that, it takes the suffix of the USERNAME in the STUN request, and looks it up in the "user frag" column of the table. If its a match, that is the user fragment needed as part of the candidate. The password is the value from that row. The sourceable transport address is also the value from that row. The priority is also copied from that row.

This new candidate can then be verified by sending STUN requests to it, as described in Section 5.3.4.

6. Running STUN on Derived Transport Addresses

One of the seemingly bizarre operations done during the ICE processing is the transmission of a STUN request to a transport address which is obtained through TURN or STUN itself. This actually does work, and in fact, has extremely useful properties. The subsections below go through the detailed operations that would occur at each point to demonstrate correctness and the properties derived from it.

6.1 STUN on a TURN Derived Transport Address

Consider a client A that is behind a NAT. It connects to a TURN server on the public side of the NAT. To do that, A binds to a local transport address, say 10.0.1.1:8866, and then sends a TURN request to the TURN server. The NAT translates the net-10 address to 192.0.2.88:5063. Assume that the TURN server is running on 192.0.2.1 and listening for TURN traffic on port 7764. The TURN server allocates a derived transport address 192.0.2.1:26524 to the client, and returns it in the TURN response. Remember that all traffic from the TURN server to the client is sent from 192.0.2.1:7764 to 10.0.1.1:8866.

Now, the client runs a STUN server on 10.0.1.1:8866, and advertises that its server actually runs on 192.0.2.1:26524. Another client, B, sends a STUN request to this server. It sends it from a local transport address, 192.0.2.77:1296. When it arrives at 192.0.2.1:26524, the TURN server "locks down" outgoing traffic, so that data packets received from A are sent to 192.0.2.77:1296. The STUN request is then forwarded to the client, sent with a source address of 192.0.2.1:7764 and a destination address of 192.0.2.88:5063. This passes through the NAT, which rewrites the source address to 10.0.1.1:8866. This arrives at A's STUN server. The server observes the source address of 192.0.2.1:7764, and generates a STUN response containing this value in the MAPPED-ADDRESS attribute. The STUN response is sent with a source address fo 10.0.1.1:8866, and a destination of 192.0.2.1:7764. This arrives at the TURN server, which, because of the lock-down, sends the STUN response with a source address of 192.0.2.1:26524 and destination of 192.0.2.77:1296, which is B's STUN client.

Now, as far as B is concerned, it has obtained a new STUN derived transport address of 192.0.2.1:7764. And indeed, it has! STUN derived transport addresses are scoped to the session, so they can only be used by the peer in the session. Furthermore, that peer has to send requests from the socket on which the STUN server was running. In this case, A is the peer, and its STUN server was on 10.0.1.1:8866. If it sends to 192.0.2.1:7764, the packet goes to the

TURN server, and due to lock-down, is forwarded to B, and specifically, is forwarded to the transport address B sent the STUN request from. Therefore, the address is indeed a valid STUN derived transport address.

The benefit of this is that it allows two clients to share the same TURN server for media traffic in both directions. With "normal" TURN usage, both clients would obtain a derived address from their own TURN servers. The result is that, for a single call, there are two bindings allocated by each side from their respective servers, and all four are used. With ICE, that drops to two bindings allocated from a single server. Of course, all four bindings are allocated initially. However, once one of the clients begins receiving media on its STUN derived address, it can deallocate its TURN resources.

[[TODO: Include a diagram that shows this pictorially.]]

6.2 STUN on a STUN Derived Transport Address

Consider a client A that is behind a NAT. It connects to a STUN server on the public side of the NAT. To do that, A binds to a local transport address, say 10.0.1.1:8866, and then sends a STUN request to the STUN server. The NAT translates the net-10 address to 192.0.2.88:5063. Assume that the STUN server is running on 192.0.2.1 and listening for STUN traffic on port 3478, the default STUN port. The STUN server sees a source IP address of 192.0.2.88:5063, and returns that to the client in the STUN response. The NAT forwards the response to the client.

Now, the client runs a STUN server on 10.0.1.1:8866, and advertises that its server actually runs on 192.0.2.88:5063. Another client, B, sends a STUN request to this address. It sends it from a local transport address, 192.0.2.77:1296. When it arrives at 192.0.2.88:5063 (on the NAT), the NAT rewrites the source address to 10.0.1.1:8866, assuming that it is of the full-cone variety [1], or is restricted, and the permission for 192.0.2.77:1296 is open. This arrives at A's STUN server. The server observes the source address of 192.0.2.77:1296, and generates a STUN response containing this value in the MAPPED-ADDRESS attribute. The STUN response is sent with a source address of 10.0.1.1:8866, and a destination of 192.0.2.77:1296. This arrives at B's STUN client.

Now, as far as B is concerned, it has obtained a new STUN derived transport address of 192.0.2.77:1296. Of course, this is the same address as the local transport address, and therefore this derived address is not used. However, had there been additonal NATs between B and A's NAT, B would end up seeing the binding allocated by that outermost NAT. The net result is that STUN requests sent to a STUN derived address behave as normal STUN would. However, these STUN requests have the side-effect of creating permissions in the NATs which see those requests in the public to private direction. This turns out to be very useful for traversing restricted NATs.

7. XML Schema for ICE Messages

This section contains the XML schema used to define the initiate, accept, and modify messages. Any protocol that uses ICE needs to map the parameters defined here into its own messages.

Note that STUN allows both the username and password to contain the space character. However, usernames and passwords used with ICE cannot contain the space.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="urn:ietf:params:xml:ns:ice"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:tns="urn:ietf:params:xml:ns:ice"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:import namespace="http://www.w3.org/XML/1998/namespace"
 schemaLocation="http://www.w3.org/2001/xml.xsd"/>
 <xs:element name="message" type="tns:message"/>
 <xs:complexType name="message">
  <xs:annotation>
   <xs:documentation>This is the root element, which holds a
             media-streams elements.</xs:documentation>
  </xs:annotation>
  <xs:sequence>
   <xs:element name="media-streams" type="tns:media-streams"/>
  </xs:sequence>
  <xs:attribute name="type" type="tns:msg-type" use="required"/>
 </xs:complexType>
 <xs:complexType name="media-streams">
  <xs:sequence>
   <xs:element name="media-stream" minOccurs="0" maxOccurs="unbounded">
    <xs:annotation>
     <xs:documentation>There are zero or more media stream
                elements. Each defines attributes for a specific media
                stream.</xs:documentation>
    </xs:annotation>
    <xs:complexType>
     <xs:sequence>
      <xs:element name="default-address">
       <xs:annotation>
        <xs:documentation>The default address is used for
                         sending media before connectivity has been
                         verified.</xs:documentation>
       </xs:annotation>
```

```
      <xs:complexType>
       <xs:complexContent>
        <xs:extension base="tns:rtp-info"/>
```

```
          </xs:complexContent>
         </xs:complexType>
        </xs:element>
        <xs:sequence>
         <xs:element name="alternate" minOccurs="0" maxOccurs="unbounded">
          <xs:annotation>
           <xs:documentation>Each alternate is a
                             possible point of contact.
                             </xs:documentation>
          </xs:annotation>
          <xs:complexType>
           <xs:complexContent>
            <xs:extension base="tns:transport-data">
             <xs:attribute name="preference" type="xs:double" use="required"/>
             <xs:attribute name="id" type="xs:string" use="required"/>
            </xs:extension>
           </xs:complexContent>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:sequence>
   </xs:complexType>
   <xs:simpleType name="msg-type">
    <xs:restriction base="xs:string">
     <xs:enumeration value="initiate"/>
     <xs:enumeration value="accept"/>
     <xs:enumeration value="modify"/>
    </xs:restriction>
   </xs:simpleType>
   <xs:complexType name="transport-data">
    <xs:sequence>
     <xs:element name="stun-user-fragment" type="xs:string"/>
     <xs:element name="stun-password" type="xs:string"/>
     <xs:element name="rtp-address" type="tns:transport-address"/>
     <xs:element name="rtcp-address" type="tns:transport-address"/>
    </xs:sequence>
   </xs:complexType>
   <xs:complexType name="transport-address">
    <xs:sequence>
     <xs:element name="ip-address" type="xs:string"/>
     <xs:element name="port">
      <xs:simpleType>
       <xs:restriction base="xs:integer">
        <xs:minInclusive value="1"/>
```

```
        <xs:maxInclusive value="65535"/>
```

```
        </xs:restriction>
       </xs:simpleType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
    <xs:complexType name="rtp-info">
     <xs:sequence>
      <xs:element name="rtp-address" type="tns:transport-address"/>
      <xs:element name="rtcp-address" type="tns:transport-address"/>
     </xs:sequence>
    </xs:complexType>
   </xs:schema>
```

8. Examples

In the examples that follow, messages are labeled with "message name A,B" to mean a message from transport address A to B. For STUN Requests, this is followed by curly brackets enclosing the username and password. For STUN responses, this is followed by square brackets and the value of MAPPED ADDRESS.

8.1 Port Restricted

This section shows a flow of two clients behind port restricted NAT talking to each other.

```
A         P.R. NAT     STUN+TURN    P.R. NAT         B
|(1) STUN Req P1,S+T      |            |             |
|----------->|            |            |             |
|            |(2) STUN Req U, S+T      |             |
|            |----------->|            |             |
|            |(3) STUN Res S+T,U [U]   |             |
|            |<-----------|            |             |
|(4) STUN Res S+T,P1 [U]  |            |             |
|<-----------|            |            |             |
|(5) Intitiate {P1,unameA,passA,q=0.4} |             |
|{U,unameA,passA,q=0.3}   |            |             |
|------------------------------------------------->|
|            |            |            |(6) STUN Req P2,S+T
|            |            |            |<-----------|
|            |            |(7) STUN Req V, S+T       |
|            |            |<-----------|             |
|            |            |(8) STUN Res S+T,V [V]    |
|            |            |----------->|             |
|            |            |            |(9) STUN Res S+T,P2 [V]
|            |            |            |----------->|
|(10) Accept {P2,unameB,passB,q=0.4}   |             |
|{V,unameB,passB,q=0.3}   |            |             |
|<-------------------------------------------------|
|(11) STUN Req P1,P2      |            |             |
|(unameBunameA,passB)     |            |             |
|----------->|            |            |             |
|            |Timeout     |            |             |
|(12) STUN Req P1,V       |            |             |
|(unameBunameA,passB)     |            |             |
|----------->|            |            |             |
```

```
      |            |(13) STUN Req U,V        |            |
      |            |(unameBunameA,passB)     |            |
      |            |------------------------>|            |
      |            |Permission open V->U     |            |
```

```
             |            |            |            |No success, Retries
continue
             |            |            |            |(14) STUN Req P2,P1
             |            |            |            |(unameAunameB,passA)
             |            |            |            |<-----------|
             |            |            |            |Timeout     |
             |            |            |            |(15) STUN Req P2,U
             |            |            |            |(unameAunameB,passA)
             |            |            |            |<-----------|
             |            |(16) STUN Req V,U        |            |
             |            |(unameAunameB,passA)     |            |
             |            |<------------------------|            |
             |            |            |            |Permission open U->V
             |            |Passes NAT! |            |            |
             |(17) STUN Req V,P1       |            |            |
             |(unameAunameB,passA)     |            |            |
             |<-----------|            |            |            |
             |(18) STUN Res P1,V [V]   |            |            |
             |----------->|            |            |            |
             |            |(19) STUN Res U,V [V]    |            |
             |            |------------------------>|            |
             |            |            |            |(20) STUN Res U,P2 [V]
             |            |            |            |----------->|
             |            |Retries continue         |            |
             |(21) STUN Req P1,V       |            |            |
             |(unameBunameA,passB)     |            |            |
             |----------->|            |            |            |
             |            |(22) STUN Req U,V        |            |
             |            |(unameBunameA,passB)     |            |
             |            |------------------------>|            |
             |            |            |            |Passes NAT! |
             |            |            |            |(23) STUN Req U,P2
             |            |            |            |(unameBunameA,passB)
             |            |            |            |----------->|
             |            |            |            |(24) STUN Res P2,U [U]
             |            |            |            |<-----------|
             |            |(25) STUN Res V,U [U]    |            |
             |            |<------------------------|            |
             |(26) STUN Res V,P1 [U]   |            |            |
             |<-----------|            |            |            |
             |(27) RTP P1,V            |            |            |
             |----------->|            |            |            |
             |            |(28) RTP U,V|            |            |
             |            |------------------------>|            |
             |            |            |            |Passes NAT! |
             |            |            |            |(29) RTP U,P2
             |            |            |            |----------->|
```

```
      |             |             |             |(30) RTP P2,U
      |             |             |             |<-----------|
```

```
          |            |(31) RTP V,U|            |            |
          |            |<------------------------|            |
          |            |Passes NAT! |            |            |
          |(32) RTP V,P1            |            |            |
          |<-----------|            |            |            |
```

9. Mapping ICE into SIP

In this section, we show how to map ICE into SIP. This requires extensions to SDP.

A new SDP attribute is defined to support ICE. It is called "alt". The alt attribute MUST be present within a media block of the SDP. It contains an alternative IP address and port (or pair of IP addresses and ports in the case of RTP) that the recipient of the SDP can use instead of the ones indicated in the m and c lines. There MAY be multiple alt attributes in a media block. In that case, each of them MUST contain a different IP address and port (or a differing pair of IP address and ports in the case of RTP).

The syntax of this attribute is:

```
alt-attribute = "alt" ":" id SP qvalue SP
                  username SP password SP
                  unicast-address SP port [unicast-address SP port]
                  ;qvalue from RFC 3261
                  ;unicast-address, port from RFC 2327
username      = non-ws-string
password      = non-ws-string
id            = token
derived-from  = ":" / id
```

With the addition of the alt attribute, the mapping of the ICE messages to SIP/SDP is straightforward. The ICE initiate message corresponds to a SIP message with an SDP offer. The ICE accept message corresponds to a SIP message with a SDP answer. The ICE modify message corresponds to a SIP INVITE or UPDATE with an offer, and the ICE modify accept message corresponds to an INVITE or UPDATE response with an answer.

Each media stream element in an ICE message maps to a media block in the SDP. The default address maps to the m and c lines in the SDP. If the ICE message indicates an RTCP address and port that are not one higher than that of the RTP, the SDP RTCP attribute [2] MUST be used to convey them.

Each alternate element in an ICE message maps either to an alt attribute in the SDP, or a new media block, depending on the IP version of the alternate. For the highest priority IPv6 alternate, it is mapped into a separate media block, using the ANAT grouping [4]. Any additional IPv6 addresses are placed as alternates within this media block. For alternates that are IPv4 addresses, the alt attribute is used. The rtp-address element maps to the first unicast-address and port components of the alt attribute. The rtcp-address element maps to the second unicast-address and port components of the alt attribute. Note that, if the RTCP address is identical to the RTP address, and the port is one higher, the second unicast-address and port MAY be omitted. The preference value from the alternate element is mapped to the q-value component of the alt attribute. The STUN user fragment and password elements map to the user fragment and password components of the alt attribute.

10. Security Considerations

ICE conveys the STUN username and password within its messages. If an eavesdropper should see the username and password, the worst they can do is send STUN requests to the host. Since STUN is a stateless protocol, the attacker can not alter the processing of the call or otherwise disrupt it. They could flood the server with BindingRequest packets. However, this would be no worse than if the attacker simply floods the host with any kind of packet.

However, integrity protection of the username and password are more important. If an attacker is capable of intercepting the message and modifying the username or password, they could prevent connectivity from being established between peers, and therefore disrupt the call. Of course, if the attacker can intercept the message, there are many other ways in which they could do that, such as simply discarding the message. Injecting fake messages with incorect usernames and passwords can also disrupt a call, and does not require the compromise of an intermediate server. A similar attack is possible by modifying most of the ICE message attributes. To prevent these kinds of attacks, it is RECOMMENDED that the actual protocols the ICE maps to make use of security mechanisms that provide message integrity protection.

11. IANA Considerations

This specification defines one new media attribute: alt. Its syntax is defined in Section 9.

12. IAB Considerations

The IAB has studied the problem of "Unilateral Self Address Fixing", which is the general process by which a client attempts to determine its address in another realm on the other side of a NAT through a collaborative protocol reflection mechanism [11]. ICE is an example of a protocol that performs this type of function. Interestingly, the process for ICE is not unilateral, but bilateral, and the difference has a signficant impact on the issues raised by IAB. The IAB has mandated that any protocols developed for this purpose document a specific set of considerations. This section meets those requirements.

12.1 Problem Definition

From RFC 3424 any UNSAF proposal must provide:

Precise definition of a specific, limited-scope problem that is to be solved with the UNSAF proposal. A short term fix should not be generalized to solve other problems; this is why "short term fixes usually aren't".

The specific problems being solved by ICE are:

Provide a means for two peers to determine the set of transport addresses which can be used for communication.

Provide a means for resolving many of the limitations of other UNSAF mechanisms by wrapping them in an additional layer of processing (the ICE methodology).

Provide a means for a client to determine an address that is reachable by another peer with which it wishes to communicate.

12.2 Exit Strategy

From RFC 3424, any UNSAF proposal must provide:

Description of an exit strategy/transition plan. The better short term fixes are the ones that will naturally see less and less use as the appropriate technology is deployed.

ICE itself doesn't easily get phased out. However, it is useful even in a globally connected Internet, to serve as a means for detecting whether a router failure has temporarily disrupted connectivity, for example. However, what ICE does is help phase out other UNSAF mechanisms. ICE effectively selects amongst those mechanisms, prioritizing ones that are better, and deprioritizing ones that are worse. Local IPv6 addresses are always the most preferred. As NATs begin to dissipate as IPv6 is introduced, derived transport addresses from other UNSAF mechanisms simply never get used, because higher priority connectivity exists. Therefore, the servers get used less and less, and can eventually be remove when their usage goes to zero.

Indeed, ICE can assist in the transition from IPv4 to IPv6. It can be used to determine whether to use IPv6 or IPv4 when two dual-stack hosts communicate with SIP (IPv6 gets used). It can also allow a client in a v6 island to communicate with a v4 host on the other side of a 6to4 NAT, by allowing the v6 host to address-fix against the v4 host, and in the process, obtain a v4 address which can be handed to the v4 client.

12.3 Brittleness Introduced by ICE

From RFC3424, any UNSAF proposal must provide:

> Discussion of specific issues that may render systems more "brittle". For example, approaches that involve using data at multiple network layers create more dependencies, increase debugging challenges, and make it harder to transition.

ICE actually removes brittleness from existing UNSAF mechanisms. In particular, traditional STUN (the usage described in RFC 3489) has several points of brittleness. One of them is the discovery process which requires a client to try and classify the type of NAT it is behind. This process is error-prone. With ICE, that discovery process is simply not used. Rather than unilaterally assessing the validity of the address, its validity is dynamically determined by measuring connectivity to a peer. The process of determining connectivity is very robust. The only potential problem is that bilaterally fixed addresses through STUN can expire if traffic does not keep them alive. However, that is substantially less brittleness than the STUN discovery mechanisms.

Another point of brittleness in STUN, TURN, and any other unilateral mechanism is its absolute reliance on an additional server. ICE makes use of a server for allocating unilateral addresses, but allows clients to directly connect if possible. Therefore, in some cases, the failure of a STUN or TURN server would still allow for a call to progress when ICE is used.

Another point of brittleness in traditional STUN is that it assumes that the STUN server is on the public Internet. Interestingly, with ICE, that is not necessary. There can be a multitude of STUN servers in a variety of address realms. ICE will discover the one that has provided a usable address.

The most troubling point of brittleness in traditional STUN is that it doesn't work in all network topologies. In cases where there is a shared NAT between each client and the STUN server, traditional STUN may not work. With ICE, that restriction can be lifted.

Traditional STUN also introduces some security considerations.

Unfortunately, since ICE still uses network resident STUN servers, those security considerations still exist.

12.4 Requirements for a Long Term Solution

From RFC 3424, any UNSAF proposal must provide:

> Identify requirements for longer term, sound technical solutions -- contribute to the process of finding the right longer term solution.

Our conclusions from STUN remain unchanged. However, we feel ICE actually helps because we believe it can be part of the long term solution.

12.5 Issues with Existing NAPT Boxes

From RFC 3424, any UNSAF proposal must provide:

> Discussion of the impact of the noted practical issues with existing, deployed NA[P]Ts and experience reports.

A number of NAT boxes are now being deployed into the market which try and provide "generic" ALG functionality. These generic ALGs hunt for IP addresses, either in text or binary form within a packet, and rewrite them if they match a binding. This will interfere with proper operation of any UNSAF mechanism, including ICE.

13. Acknowledgements

The authors would like to thank Douglas Otis and Francois Audet for their comments and input.

14. References

14.1 Normative References

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

[2] Huitema, C., "Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)", RFC 3605, October 2003.

[3] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[4] Camarillo, G., "The Alternative Network Address Types Semantics for the Session Description Protocol Grouping Framework", draft-ietf-mmusic-anat-01 (work in progress), June 2004.

14.2 Informative References

[5] Schulzrinne, H., Rao, A. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[6] Senie, D., "Network Address Translator (NAT)-Friendly Application Design Guidelines", RFC 3235, January 2002.

[7] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A. and A. Rayhan, "Middlebox communication architecture and framework", RFC 3303, August 2002.

[8] Borella, M., Lo, J., Grabelsky, D. and G. Montenegro, "Realm Specific IP: Framework", RFC 3102, October 2001.

[9] Borella, M., Grabelsky, D., Lo, J. and K. Taniguchi, "Realm Specific IP: Protocol Specification", RFC 3103, October 2001.

[10] Yon, D., "Connection-Oriented Media Transport in the Session Description Protocol (SDP)", draft-ietf-mmusic-sdp-comedia-07 (work in progress), June 2004.

[11] Daigle, L. and IAB, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation", RFC 3424, November 2002.

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

[13] Baugher, M., McGrew, D., Naslund, M., Carrara, E. and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[14] Carpenter, B. and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", RFC 3056, February 2001.

[15] Huitema, C., "Teredo: Tunneling IPv6 over UDP through NATs", draft-huitema-v6ops-teredo-02 (work in progress), June 2004.

[16] Rosenberg, J., "Traversal Using Relay NAT (TURN)", draft-rosenberg-midcom-turn-04 (work in progress), February 2004.

Author's Address

Jonathan Rosenberg
dynamicsoft
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
EMail: jdrosen@dynamicsoft.com
URI: http://www.jdrosen.net

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.