

Interactive Connectivity Establishment (ICE): A Methodology for
Network Address Translator (NAT) Traversal for Multimedia Session
Establishment Protocols
[draft-ietf-mmusic-ice-03](#)

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 April 25, 2005.

Copyright Notice

Copyright (C) The Internet Society (2004). All Rights Reserved.

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	3
2.	Multimedia Signaling Protocol Abstraction	4
3.	Terminology	5
4.	Overview of ICE	7
5.	Detailed ICE Algorithm	9
5.1	Initiator Processing	10
5.1.1	Sending the Initiate Message	10
5.1.2	Processing the Accept	11
5.2	Responder Processing	11
5.2.1	Processing the Initiate Message	11
5.3	Common Procedures	12
5.3.1	Gathering Transport Addresses	12
5.3.2	Enabling STUN on Each Local Transport Address	14
5.3.3	Prioritizing the Transport Addresses and Choosing a Default	16
5.3.4	Sending STUN Connectivity Checks	18
5.3.5	Receiving STUN Requests	23
5.3.6	Management of Resources	24
5.3.7	Binding Keepalives	24
6.	Running STUN on Derived Transport Addresses	25
6.1	STUN on a TURN Derived Transport Address	26
6.2	STUN on a STUN Derived Transport Address	28
7.	XML Schema for ICE Messages	29
8.	Example	31
9.	Mapping ICE into SIP	34
9.1	Message Mapping	34
9.2	SIP and SDP Specific Security Considerations	36
9.3	Updates in the Offer/Answer Model	36
10.	Security Considerations	36
11.	IANA Considerations	37
11.1	SDP Attribute Name	37
11.2	URN Sub-Namespace Registration	38
11.3	XML Schema Registration	39
12.	IAB Considerations	39
12.1	Problem Definition	40
12.2	Exit Strategy	40
12.3	Brittleness Introduced by ICE	41
12.4	Requirements for a Long Term Solution	41
12.5	Issues with Existing NAPT Boxes	42
13.	Acknowledgements	42
14.	References	42
14.1	Normative References	42
14.2	Informative References	43
	Author's Address	44
	Intellectual Property and Copyright Statements	45

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) [9] 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 [10]. 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 [11], Simple Traversal of UDP through NAT (STUN) [1], Traversal Using Relay NAT [8], and Realm Specific IP [12][13] along with session description extensions needed to make them work, such as the SDP attribute for RTCP [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 define an abstraction of a multimedia signaling system, and define terms that can be used throughout this specification. Figure 1 shows the abstraction.

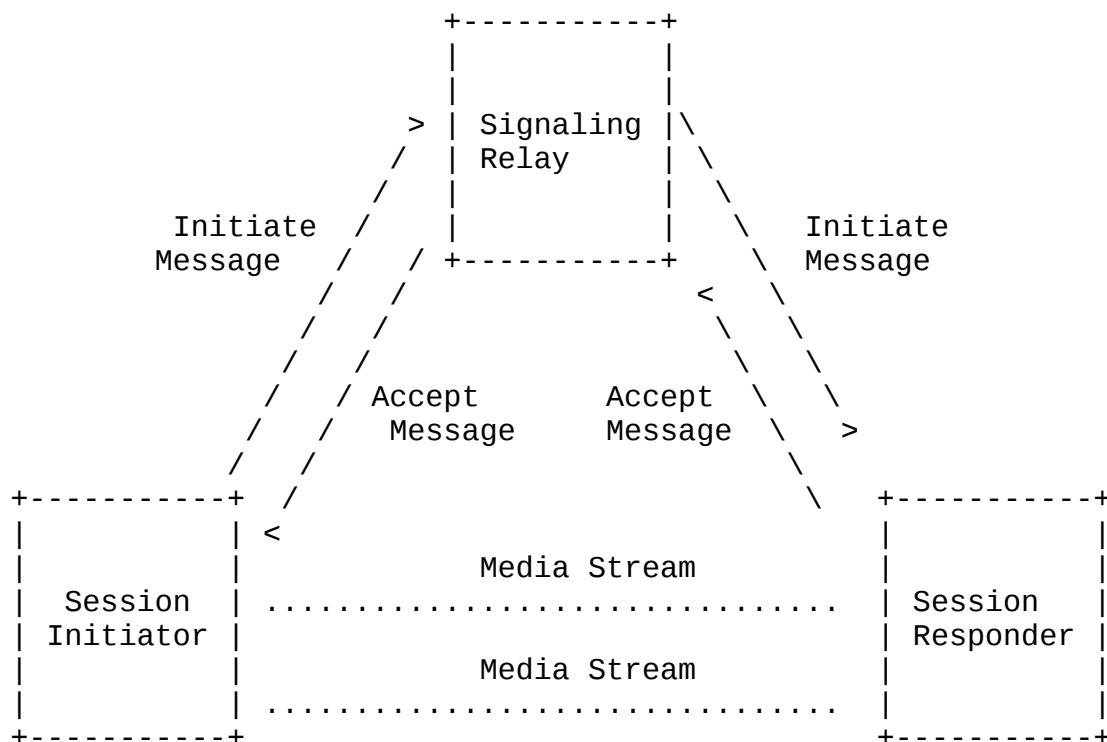


Figure 1

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 the user agent that generates an SDP offer [4], the responder is a SIP user agent that generates an SDP answer to the offer, 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.

The initiate and accept messages need to contain parameters, defined by this specification, for the protocol to operate. The initiate and accept messages are therefore defined by this specification as XML documents containing the relevant information. Of course, multimedia signaling protocols will not use these XML documents directly. Rather, those protocols will need to define extensions as needed to show how the initiate, accept and terminate messages map to messages in the actual protocol, and how every element and attribute in the XML document for those messages maps into parameters of the actual protocol. [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. For SIP, this is any SIP message that contains an offer. Usually, this is the initial INVITE.

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. For SIP, this is any SIP message that contains an answer. Usually, this is a 200 OK.

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 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) [[12](#)] (which lives at the operating system level). Transport addresses are typically obtained by binding to an interface.

Usable Local Transport Address: A local transport address created for the purposes of advertisement to ICE peers.

Associated Local Transport Address: An associated transport address is a local transport address used solely to obtain a derived transport address. Associated local transport addresses are never advertised in ICE messages. However, packets are received on them when sent to the derived transport address.

Derived Transport Address: A derived transport address is a transport address which is derived from an associated local transport address. The derived transport address is related to the associated local transport address in that packets sent to the derived transport address are received on the socket bound to its associated local transport address. Derived addresses are

obtained using protocols like STUN and TURN, and more generally, any UNSAF protocol [[14](#)].

Advertised Transport Addresses: The union of the usable local transport addresses and the derived transport addresses. These are the ones used in ICE messages.

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.

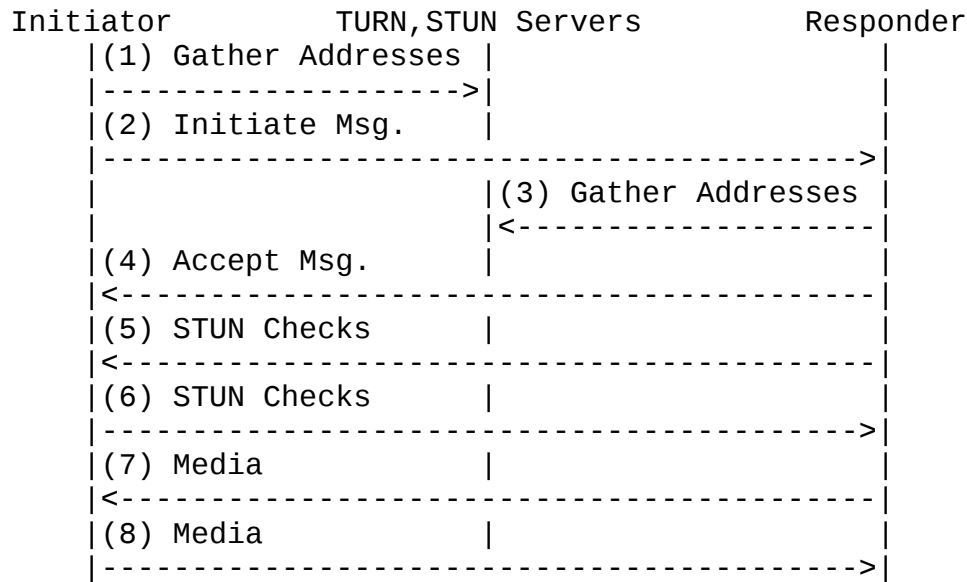


Figure 2

The basic flow of operation for ICE is shown in Figure 2. Before the initiator establishes a session, it obtains as many IP address and port combinations in as many address realms as it can. These addresses 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 the local transport addresses it has obtained. These include ones that will be advertised directly through ICE, and so-called associated local transport addresses, which are not directly advertised; rather, the transport address derived from them is advertised. 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 one half of the STUN username and the 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, along with the other half of the STUN username and its password, 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 initiator. Once the initiator 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 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.

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, both usable and associated, 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. Doing so eliminates any additional perceivable call setup delays due to address gathering.

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 an IPv4 and/or an IPv6 default address, and a list of candidates. Each candidate has information for RTP and optionally RTCP. RTCP information is optional since, unfortunately, many systems don't support it. If ICE did not indicate that RTCP was not supported, connectivity checks would be made to the RTCP ports and fail, confusing operation and adding unnecessary overhead.

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 candidates represent addresses that the responder should try using the mechanisms of this specification. The list of candidates includes the defaults. In SIP, the candidates are conveyed with the new SDP candidate parameter.

The client then encodes its usable local transport addresses and derived transport addresses (including the one set as the default) as a series of candidate elements. Each candidate element conveys a transport address for RTP, a transport address for RTCP, a STUN username fragment and STUN password for RTP, and one for RTCP. The client MUST assign each candidate a unique identifier. These identifiers MUST be unique across all candidates used within the session. Though they are not used in this specification, they serve as a convenient and short handle for each candidate within the document. Experience has shown that explicit identifiers for elements in SDP is a good idea. This identifier is encoded in the "id" attribute of the <candidate> 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 as the destination. 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 through the TURN server, which will then forward them towards the responder.

If the Accept message contains candidates, it implies that the responder supported ICE. In that case, the initiator takes each candidate transport address, STUN username fragment, 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. By gathering addresses while the user is being alerted to the request for communications, session establishment delays due to that gathering can be eliminated.

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 usable local transport address and derived transport address it has gathered, and identifies one as a default, as described in [Section 5.3.3](#).

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

The accept message 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 - usable local transport addresses and derived transport addresses. Usable 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) [[15](#)], the client will need to bind to an ephemeral port for both RTP and RTCP.

The result will be a set of usable local transport addresses. The client may also have access to servers that provide unilateral self-address fixing (UNSAF) [[14](#)]. Examples of such protocols include STUN, TURN, and TEREDO [[18](#)]. UNSAF protocols work by having the client send, from a specific associated 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 associated 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 associated local transport address.

All ICE implementations SHOULD implement and use STUN and TURN for unilateral allocation. STUN is an integral part of this specification for connectivity checks and will always be present for

that purpose. The usage of TURN and STUN for unilateral allocations is at SHOULD strength, and not MUST, since there are many network environments, and there will be deployments for which one of these will never be used and will impose needless cost. However, one of the key ideas behind ICE is that network conditions and connectivity assumptions can, and will change. Just because a client is communicating with a server on the public network today, doesn't mean that it won't need to communicate with one behind a NAT tomorrow. Just because a client is behind a full cone NAT today, doesn't mean that tomorrow they won't pick up their client and take it to a public network access point where there is a symmetric NAT. The way to handle these cases and build a reliable system is for clients to implement a diverse set of techniques for allocating addresses, so that at least one of them is almost certainly going to work in any situation. The combination of TURN, STUN and local address allocations provide sufficient coverage to handle nearly any NAT configuration. Implementors should consider very carefully any assumptions that they make about deployments before electing not to implement one of these mechanisms for address allocation. In particular, implementors should consider whether the elements in the system may be mobile, and connect through different networks with different connectivity. They should also consider whether endpoints which are under their control, in terms of location and network connectivity, would always be under their control. Only in cases where implementors truly believe that these cases will not require either TURN or STUN allocations, should those techniques not be implemented.

For each UNSAF protocol, 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 server for each UNSAF protocol, the client MUST bind to a new local transport address, and uses it to obtain a single derived transport address for it. This local IP address and port is called an associated transport address. These addresses are not advertised to peers in ICE messages; their derived transport addresses are. As a result of using a different local transport address for each derived transport address, every transport address advertised in an ICE message is either a unique local transport address, or else is derived from a unique local transport address.

If a derived transport address is equal to the associated local transport address from which it was derived, the local transport address SHOULD be promoted to a usable local transport address. It is preferable to do this than to use a new local transport address; the UNSAF protocol may have caused pinholes to open in intervening firewalls.

Implementations MAY use other protocols that provide derived transport addresses, as long as those techniques meet the following conditions:

1. The technique does not require its peer to know about, or understand the technique in order to interoperate.
2. The technique can provide the client with an IP address and port that may be reachable by some peers.
3. The technique allows the client to receive STUN connectivity checks in addition to media packets on the same IP address and port.
4. The technique allows the client to send packets to a peer, so that the peer will see the derived transport address as the source IP address and port of the packet.

[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 both associated local transport addresses and usable transport addresses. These include ones used for both RTP and 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. The CHANGED-ADDRESS attribute in a BindingResponse is set to the transport address on which the server is running.

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 [\[16\]](#), 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 transport address advertised in the initiate message, the client MUST choose a username fragment and a password. The username fragment created by the client (called the local username fragment) is concatenated with the fragment created by its peer (called the remote username fragment) to create the actual username used for access to the STUN server that will receive packets sent to that transport address. This username will be present in STUN requests sent by its peer. 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 with high probability, and different for each advertised transport address. It SHOULD be persistently used over time for that particular transport address. A value computed as the 128 bit hash of the transport address concatenated with a 128 bit random number selected to identify the host will meet these requirements. This results in two properties. First - each transport address can be uniquely identified. Secondly, no other host will select a username with the same value. The password MUST be random with at least 128 bits of randomness and is selected separately for each transport address advertised as part of a distinct session. This means that RTP and RTCP, which run on different transport addresses, will get different usernames and passwords. The password will remain constant during a session with a peer, but will otherwise vary across sessions. The username fragment and password will be passed to its peer in an initiate or accept message. Because the password is conveyed through these signaling protocols, those protocols MUST provide facilities for encryption, authentication and message integrity, and those facilities SHOULD be used when ICE is employed. 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 uniqueness

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, that's 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 fragment takes on the role of a unique identifier. C provides A with a unique username fragment, and A provides one to C. A uses these two fragments to construct the 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.

An unfortunate consequence of the non-uniqueness of IP addresses is that, in the above example, B might not even be an ICE client. It could be any host, and the port to which the STUN packet is directed could be any ephemeral port on that host. If there is an application listening on this socket for packets, and it is not prepared to handle malformed packets for whatever protocol is in use, the operation of that application could be effected. Fortunately, since the ports exchanged in SDP are ephemeral and usually drawn from the dynamic or registered range, the odds are good that the port is not used to run a server on host B, but rather is the client side of some protocol. This decreases the probability of hitting a port in-use, due to the transient nature of port usage in this range. However, the possibility of a problem does exist, and network deployers should be prepared for it.

Termination of the local STUN servers is discussed in [Section 5.3.6](#).

[5.3.3](#) Prioritizing the Transport Addresses and Choosing a Default

The prioritization process takes the list of the advertised 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 do 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) [17]. 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 addresses 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 corporate 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. Unfortunately, it is difficult to ascertain which address that might be. As an example, consider a user within an enterprise. To reach non-ICE capable clients within the enterprise, a local transport address has to be used, since the enterprise policies may prevent communication between elements using a relay on the public network. However, when communicating to peers outside of the enterprise, a TURN-based public address is needed.

Indeed, the difficulty in picking just one address that will work is the whole problem that motivated the development of this specification in the first place. As such, it is RECOMMENDED that the default address be a TURN derived transport address from a TURN server providing public IP addresses. Furthermore, ICE is only truly effective when it is supported on both sides of the session. It is therefore most prudent to deploy it to close-knit communities as a whole, rather than piecemeal. In the example above, this would mean that ICE would ideally be deployed completely within the enterprise, rather than just to parts of it.

[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 from the peer (called the remote username fragment), the password from the peer (called the remote password), and priority from the peer (called the remote priority) 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. An example of a candidate table for RTP audio is shown below.

Remote Transport Address	Remote Username Fragment	Remote Password	Remote Priority

10.0.1.1:38746	asd9f8f8==	9asfhfvva9==affahnz	0.4
192.0.2.77:44634	xcyca87sbb	f99fhaz0ftrafdgl99d	0.2

Figure 3

The client then creates a new table, called the connection table. There is a row in this table for each gathered address and remote transport address pair. This table has a column for the local transport address, which is equal to the gathered address if it was a usable local transport address, else equal to the associated local transport address if the gathered address was a derived address. There is also a column for the remote transport address, the local username fragment, the remote username fragment, the remote password and the state. Each row in this table is called a connection, and it provides information on the connectivity when sending packets from the local transport address to the remote transport address.

There are four possible states for each connection. These states are:

INIT: No STUN transaction has been completed towards this remote transport address from this local transport 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 local transport address. This state is important for connectivity checks made to STUN derived transport addresses through port restricted NAT.

BAD: All STUN transactions to this remote transport address from this local transport 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 local transport address.

GOOD: A STUN transaction to this remote transport address from this local transport address was successful.

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

Consider the the following example. An initiator sends an initiate message with one media stream (audio), with two RTP transport addresses, 10.0.1.1:38746 (which we denote "A" for shorthand) and 192.0.2.77:44634 (which we denote "B" for shorthand). A is a usable local transport address, and B is a STUN derived transport address (although that fact is not signaled in the message). The usernames and passwords for these transport addresses are shown in Figure 3. The initiate message is sent to the responder. The responder has a local transport address (10.0.1.76:43443), and a a STUN derived transport address (192.0.2.64:54766) derived from (10.0.1.76:43444). Call these two local transport addresses X and Y respectively. The

connection table created by the responder would have four rows (two local transport addresses times two remote transport addresses). Such a table might look like this:

Remote Trans. Address	Local Trans. Address	Remote Username Fragment	Local Username Fragment	Remote Password	Remote Priority	State
A	X	asd9f8f8==	8asd77fa9	9asfhfvva9==affahnz	0.4	INIT
A	Y	asd9f8f8==	zhff8dga^	9asfhfvva9==affahnz	0.4	INIT
B	X	xcyca87sbb	8asd77fa9	f99fhaz0ftrafdgl99d	0.2	INIT
B	Y	xcyca87sbb	zhff8dga^	f99fhaz0ftrafdgl99d	0.2	INIT

The client begins a STUN BindingRequest transaction for each connection. This STUN transaction is sent to the IP address and port from the Remote Transport Address column. It sends the request from the IP address and port in the Local Transport Address column. The STUN USERNAME attribute MUST be present. It is set to the concatenation of the remote user fragment with the local user fragment from the table. Thus, for the candidate with remote transport address A and local transport address X, the USERNAME would be set to "asd9f8f8==8asd77fa9". 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 a 420, 500, or 401, the client should try again as described in [RFC 3489](#). Either initially, or after such a retry, the STUN transaction will produce a timeout result, a success result, a fundamentally non-recoverable failure result (error codes 400, 431, or 600) or a failure result inapplicable to this usage of STUN and thus unrecoverable (432, 433), or a 430 error. These correspond to the "timeout", "success", "error" and "race-failure" events, respectively. The 430 response code, as described below, is generated when the server doesn't recognize the STUN username, presumably because the BindingRequest was sent to the initiator prior to receipt of the ICE Accept message by the initiator. Its occurrence is thus a result of a failed race between the BindingRequest and Accept message. As the state machine below discusses, the client will retry in this case.

These events are fed into the finite state machine (FSM) described in Figure 5. This figure shows the transitions between states that

occur on the completion of the STUN BindingRequest transaction or upon the expiration of timers set by the FSM.

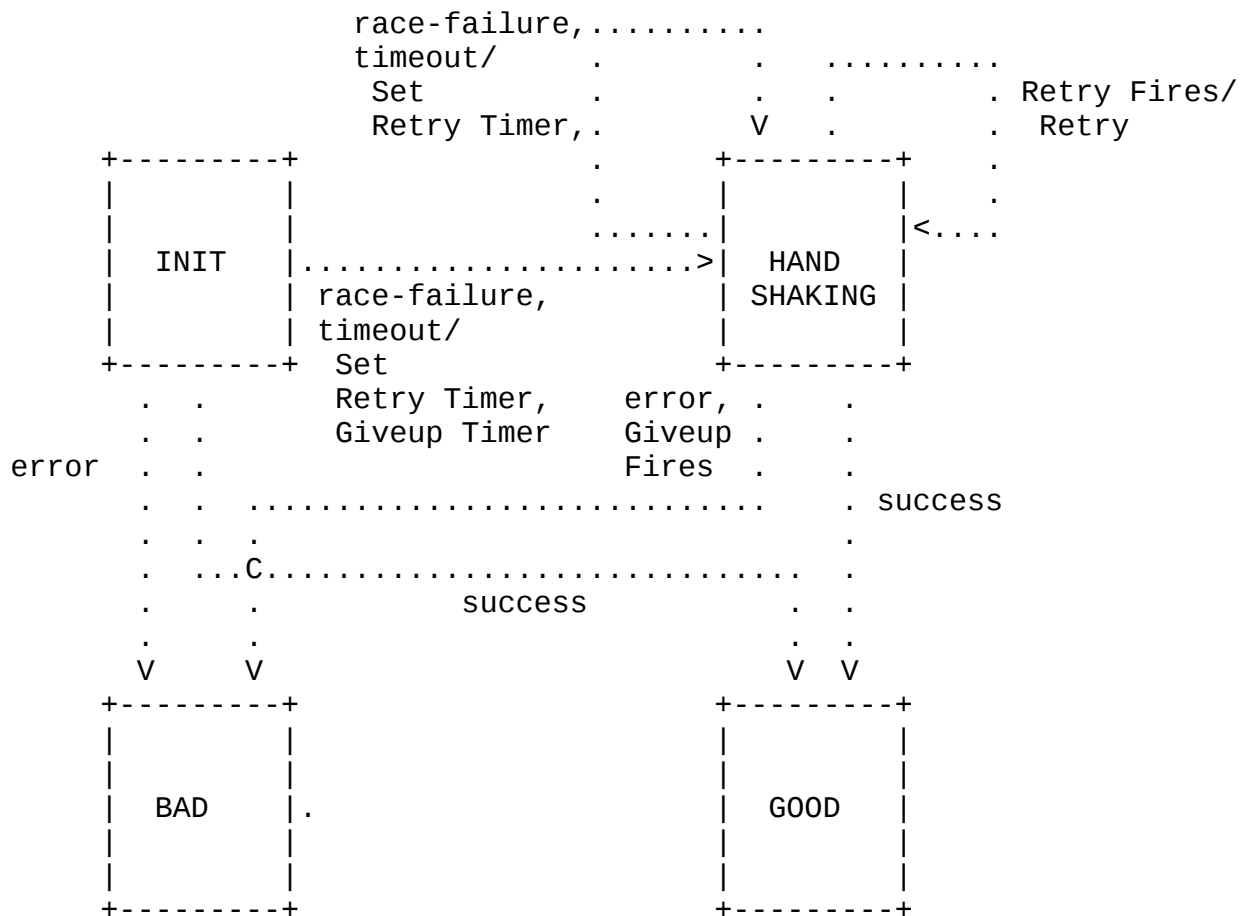


Figure 5

Starting in the INIT state, if the transaction is successful, the client has verified connectivity to that remote transport address when sending from that local 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. If, from the INIT state, the STUN transaction times out, the FSM enters the HANDSHAKING state. At this point, there are two likely reasons that the STUN transaction 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. It is also possible that the STUN transaction timed out due to a persistent network failure, in which case, a retry is in order. As such, the HANDSHAKING 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 HANDSHAKING 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 50ms. Note that this timer does not mean that a STUN request is repeated every 50ms. It means that a new STUN transaction begins 50ms after the completion of the previous one. STUN transactions themselves employ exponentially back off retransmit timers. 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 a race-failure event, it means that the peer has not yet completed the initiate/accept exchange, and thus the username has not been allocated. Another BindingRequest transaction needs to take place to try again. Thus, the same retry and giveup timers as in the timeout event are started.

If, from the INIT state, the STUN transaction generates an error, the FSM moves into the BAD state. This state means that the connection is definitively unreachable, and it will not be used subsequently in the session.

If, while in the HANDSHAKING state, the Giveup timer fires, or the STUN transaction results in an error, the client moves into the BAD state. If, while in the HANDSHAKING state, the Rapid Retry timer fires, a new STUN transaction is started. The output of that transaction will be subsequently fed into the FSM, but upon initiation of the retry attempt there is no change in state. If the pending BindingRequest transaction succeeds, the FSM moves into the GOOD state. This transport connection is viable for communications.

Once one of the connections in the connection table enters the GOOD state, the client SHOULD begin using it for communications. It SHOULD cease any ongoing transactions and terminate FSMs for connections of lower priority. If, another connection of higher priority should subsequently enter the GOOD state, the client SHOULD switch to that one, and once more cease all ongoing transactions and terminate FSMs for connections of lower priority. It SHOULD perform this switch after waiting a small period of time (2 seconds is

RECOMMENDED) to prevent against quick changes in transport address as each of the ongoing connectivity checks completes. If there are multiple GOOD connections whose priorities are equal and higher than any other GOOD connection, the client SHOULD pick one randomly and use that. It SHOULD NOT change to another one of equal priority later on. Each change in address is likely to cause a change in transport characteristics, and manifest itself as a "glitch" to the user.

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

[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 "local username fragment" column in the connection table. If it doesn't match any rows, the client generates a 400 response. If it matches one or more rows, the client checks the suffix of the username against the "remote username fragment" column in those matching rows. If the final result doesn't match any rows, the client generates a 430 response. If the final result matches a single row, that row identifies the connection on which the STUN request was received. The client then proceeds with the processing of the request and generation of a response as 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 of the matching connection from the previous paragraph. If they don't match, and that remote transport address is not elsewhere in the table, this source transport address is itself another possible candidate. As with other candidates, it must be associated with a STUN remote username fragment, remote password and remote priority. These are obtained from the values of these columns for the matching connection in the table. This candidate is then paired with each local transport address, and the resulting set of connections are added to the connection table and verified using STUN connectivity checks as per [Section 5.3.4](#).

When will the source transport address of the BindingRequest not match an existing candidate remote transport address? This happens when there is a NAT between the peers which is not on the path

between each peer and the UNSAF servers.

5.3.6 Management of Resources

The beginning of a multimedia session results in the creation of several resources to support ICE. These include gathered addresses, both local and derived, along with the local STUN servers that run on the local addresses. These resources must be maintained and eventually freed.

It is RECOMMENDED that all gathered addresses be retained for the duration of the session. Even if they are not used initially, this allows them to be used later in the session should conditions change, requiring a signaling operation to update the set of candidate addresses. Maintaining these resources depends on the type of resource. For a local transport address, nothing is required. The socket is maintained until freed by the ICE application. For STUN derived transport addresses, the bindings in the NAT for that address need to be maintained. If the derived transport address is used by the peer for media, the media itself serves to keep the bindings alive (see [Section 5.3.7](#)). A client can determine that a STUN derived transport address was used for media when the RTP packet arrives at the associated local transport address. For the other STUN derived transport addresses, the client SHOULD periodically generate STUN transactions to the STUN server. Every 20 seconds is RECOMMENDED.

For TURN derived transport addresses, the bindings in the NAT along with the mappings in the TURN server need to be maintained. Media traffic itself can accomplish that. The client will know that its TURN derived transport address is in use when an RTP packet arrives at the associated local transport address. For other TURN derived transport addresses, the TURN keepalive mechanisms SHOULD be used.

Once the STUN servers are started on the local transport addresses, they MUST run until a valid media packet is detected on that transport address. Once a media packet is received, it signals that the peer has completed its connectivity checks and has decided to use that transport address (or the derived transport address, as the case may be) for media communications. 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](#)

5.3.7 Binding Keepalives

Once the STUN connectivity checks complete, STUN packets are no longer used. However, bindings in intermediate NATs need to be kept alive so that the media can continue to flow. Doing so is the

responsibility of the media protocol.

In the case of RTP, the RTP packets themselves normally come sufficiently quickly to keep the bindings alive. However, several cases merit further discussion. Firstly, in some RTP usages, such as SIP, the media streams can be "put on hold". This is accomplished by using the SDP "sendonly" or "inactive" attributes, as defined in [RFC 3264](#) [4]. [RFC 3264](#) directs implementations to cease transmission of media in these cases. However, doing so may cause NAT bindings to timeout, and media won't be able to come off hold.

As such, clients SHOULD instead send a media packet periodically, independent of whether the stream is "sendonly", "recvonly" or "inactive". At least once every 20 seconds is RECOMMENDED. These packets can be sent using any of the payload formats listed by the peer in its SDP. For audio streams, It is RECOMMENDED that implementations support the RTP payload format for comfort noise [5], which makes a good choice. For video codecs, a minimally coded frame is a good choice.

Secondly, some RTP payload formats, such as the payload format for text conversation [19], may send packets so infrequently that the interval exceeds the NAT binding timeouts. In such cases, the implementation should send some any kind of content, if possible. If the payload type doesn't allow anything meaningful to be sent, even a malformed RTP packet is superior to nothing at all; the malformed packet would be rejected by the peer, and have the side effect of keeping the NAT bindings open.

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. They are tutorial in nature.

6.1 STUN on a TURN Derived Transport Address

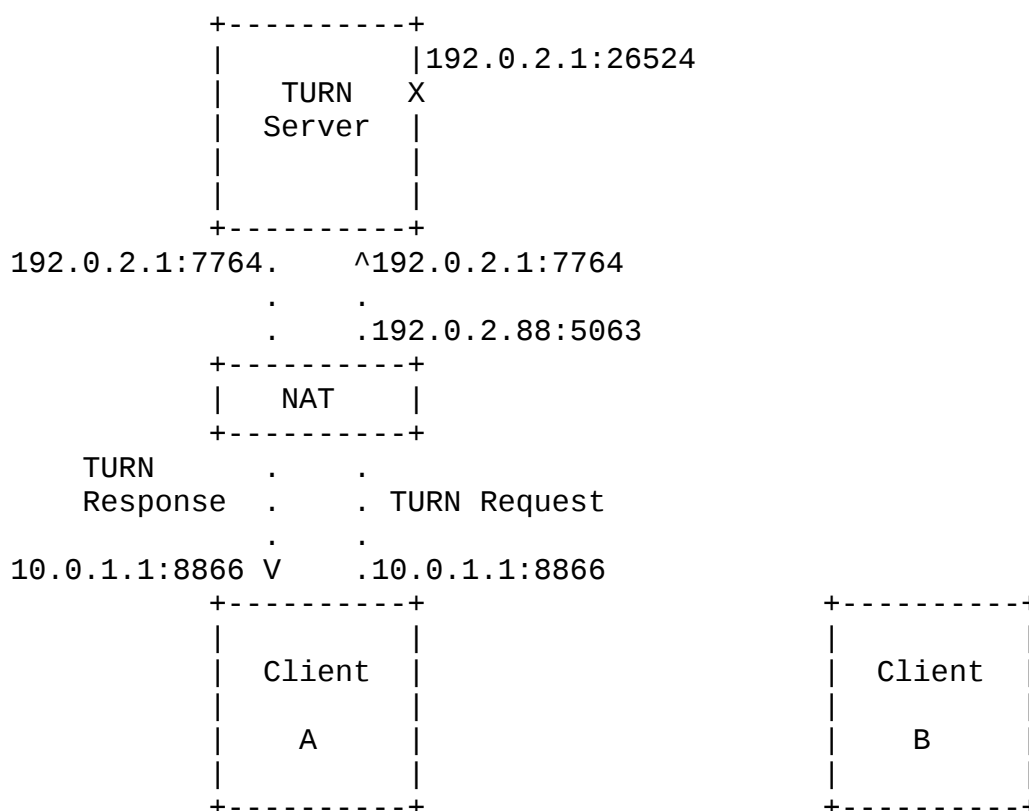


Figure 6

Consider a client A that is behind a NAT, shown in Figure 6. 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 (shown as the X on the TURN server in the diagram), 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, including the TURN response.

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, it is discarded since client A has not sent a packet to 192.0.2.77:1296. Once client A gets client B's accept message, it

will learn about B's candidate address, and generate a STUN request towards it. This results in a permission being installed in the TURN server, so that packets from 192.0.2.77:1296 will now be accepted. The next STUN request from client B will therefore succeed. This is the normal mode of operations for port restricted NAT; as described in TURN, the server turns a symmetric NAT into a port restricted one [8].

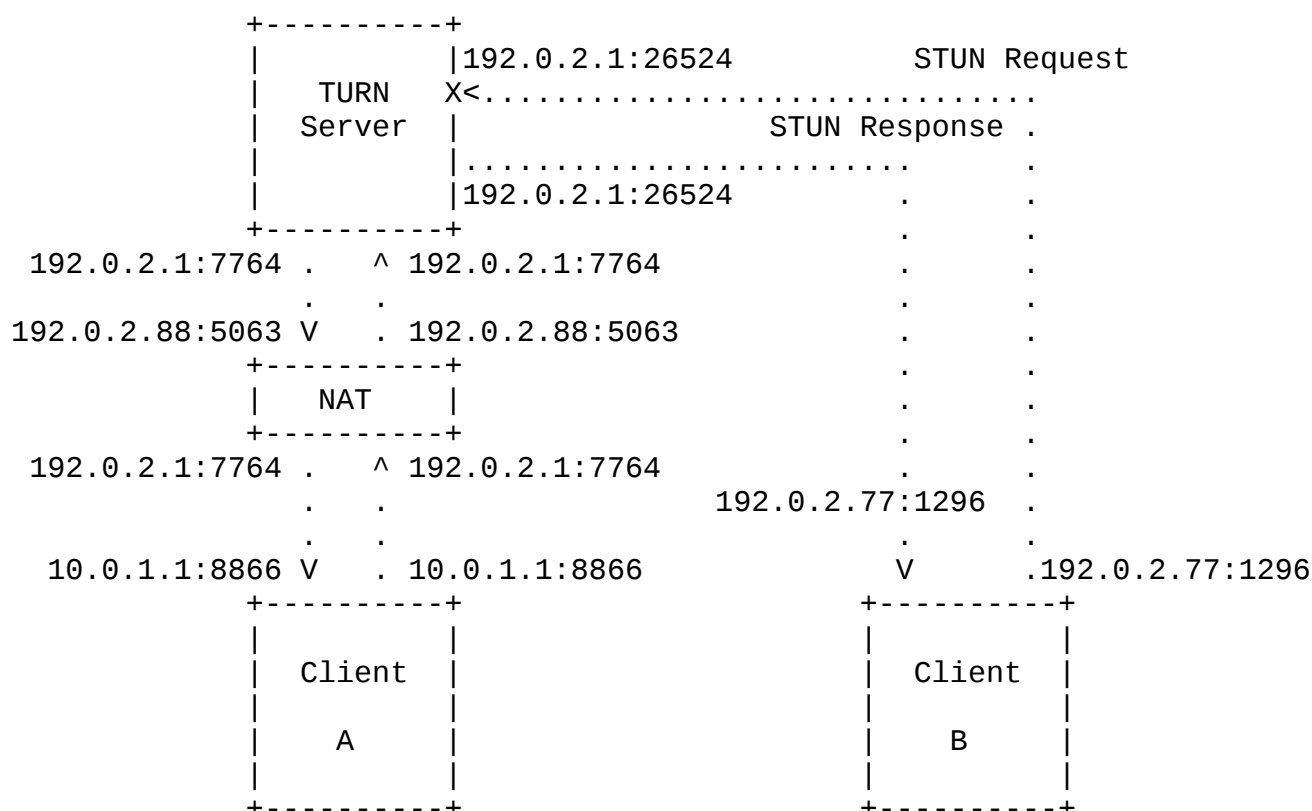


Figure 7

As shown in Figure 7, client B will retry, sending it STUN request from 192.0.2.77:1296 to 192.0.2.1:26524. This successful STUN request is 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 destination 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 of 10.0.1.1:8866, and a destination of 192.0.2.1:7764. This arrives at the TURN server, which, because of current destination is 192.0.2.1:7764, 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 A is concerned, it has obtained a new candidate 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 since the destination address is set to 192.0.2.77:1296, 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 candidate transport address. Its priority is derived from the priority of client B's public IP 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.

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 transport address is 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 or restricted variety [1], and the permission for 192.0.2.77:1296 is open. This arrives at A's local 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 A is concerned, the STUN request had a source transport address which was already known to A, presumably from an ICE exchange. As far as B is concerned, the check succeeded, and the address is viable.

7. XML Schema for ICE Messages

This section contains the XML schema used to define the initiate and accept 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:tns="urn:ietf:params:xml:ns:ice"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  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:element>
    </xs:sequence>
  </xs:complexType>
  <xs:sequence>
    <xs:element name="default-addresses">
```

```

    <xs:complexType>
      <xs:sequence>
        <xs:element name="ipv4-address" type="tns:rtp-info" minOccurs="0"/>
        <xs:element name="ipv6-address" type="tns:rtp-info" minOccurs="0"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:sequence>
    <xs:element name="candidate" minOccurs="0" maxOccurs="unbounded">
      <xs:annotation>
        <xs:documentation>Each candidate is a possible point
of media reception.</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:complexType>
</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="rtp-address" type="tns:transport-address"/>
    <xs:element name="rtcp-address" type="tns:transport-address" minOccurs="0"/>
    <xs:element name="rtp-stun-info" type="tns:stun-info"/>
    <xs:element name="rtcp-stun-info" type="tns:stun-info" minOccurs="0"/>
  </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>

```

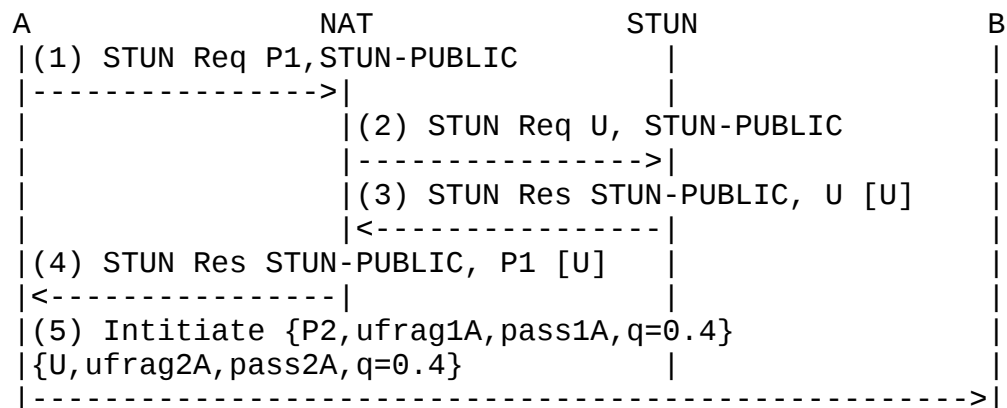
```

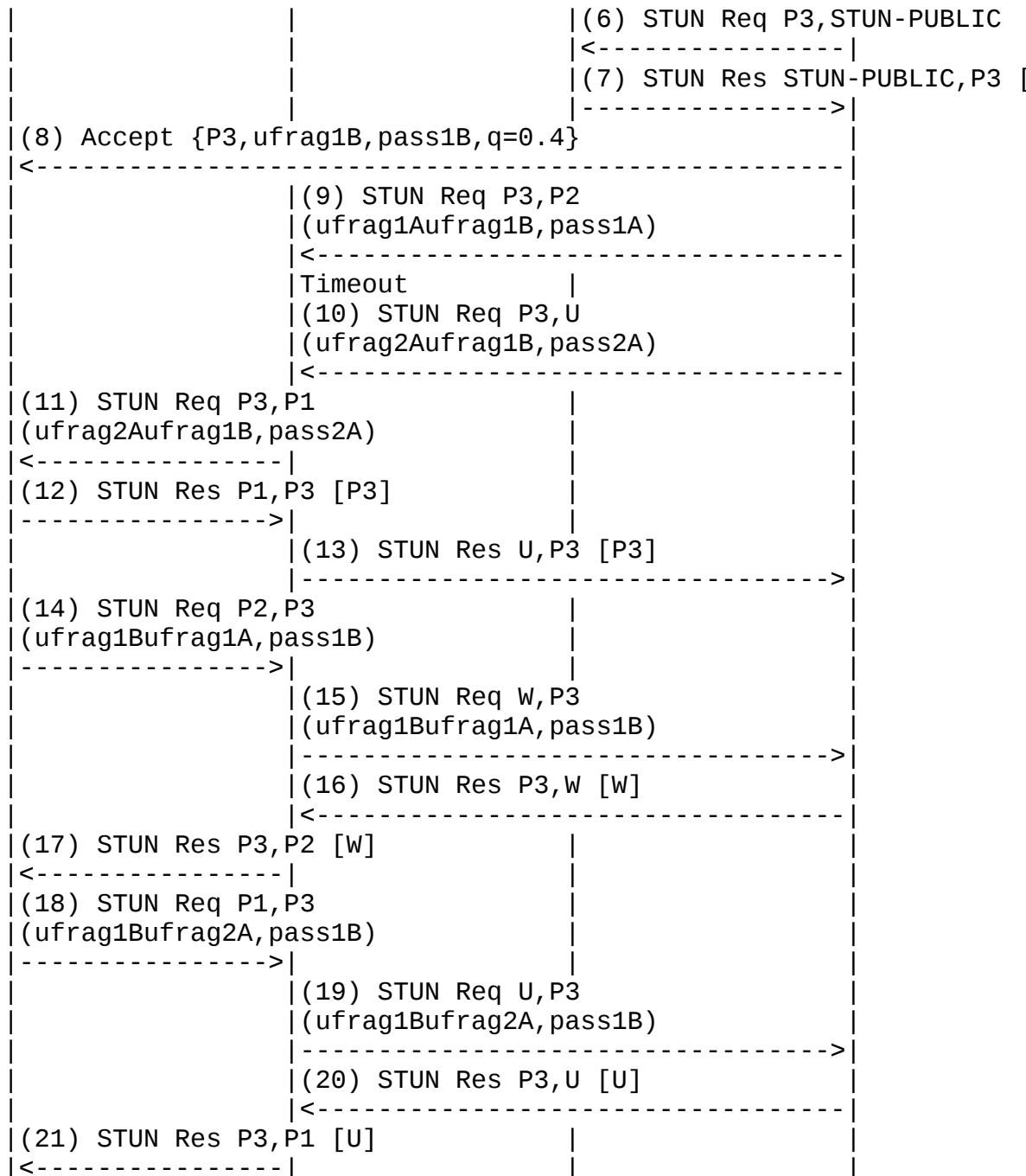
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="1"/>
      <xs:maxInclusive value="65535"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:complexType name="stun-info">
  <xs:sequence>
    <xs:element name="username-fragment"/>
    <xs:element name="password"/>
  </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"
      minOccurs="0"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>

```

8. Example

In the example that follows, 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. The example shows a flow of two clients where one is behind a full cone NAT, and the other is on the public Internet.





The initiator, client A, binds to a local transport address P1, which will be used as an associated local transport address. As such, it sends a STUN request to its STUN server (message 1). This passes through a NAT, and the NAT maps private address P1 to public address U (message 2). The STUN server mirrors this public address in the MAPPED-ADDRESS of the STUN response (message 3), and it is forwarded

to the initiator (message 4). Now, client A has a STUN derived transport address of U. It also binds to a second local transport address, P2, which will be a usable local transport address. It starts STUN servers on both local transport addresses P1 and P2. It then generates an Initiate request to client B (message 5) which contains both of the gathered transport addresses P2 and U, along with username fragments and passwords.

Client B is not behind a NAT. It binds to a local transport address P3, and sends a STUN request to its STUN server (message 6). This is responded to by the STUN server (message 7). The client observes that this address is identical to its local transport address, and therefore that local transport address is, which was targeted for an associated local transport address, is promoted to a usable local transport address. It then sends an Accept message to client A, including this transport address and its username fragment and password (message 8).

Once the Accept message is sent, the client can perform its STUN connectivity checks. B has a single local transport address (P3), which it matches up with A's two remote transport addresses (P2 and U). B tries P2 (message 9). This request fails since P2 is a private address. In parallel, B tries U (message 10). Since A's NAT is full cone, this packet is accepted and is passed to client A (message 11). Client A generates a response (message 12) which is forwarded to client B (message 13). The source transport address in the STUN packet, P3, is already known to client A, and thus no new candidates are learned. Client B learns that client A is reachable at transport address U, but not P3. Thus, it can begin sending media to U from local transport address P3.

Once the Accept message arrives at client A, it can begin its connectivity checks. It has two local transport addresses P1 and P2, which it combines with client B's single transport address P3. It tries to send a STUN packet from P2 to P3 (message 14). Since the NAT has not seen source address P2 yet, it maps it to a new public transport address W, and the STUN request is forwarded to client B (message 15). Client B generates a STUN response (message 16), which is forwarded back to client A (message 17). Based on this, client A learns that it can reach P3 from P2. Client B learns a new remote transport address, W. However, the priority of this address is the same as P2, which is 0.4, and equal to the priority of address U, to which client B has already connected. Thus, it does not bother to perform the check (such a check would have succeeded if it had been done).

While the P2->P3 check is taking place, client A also sends a STUN request from P1 to P3 (message 18). This passes through the NAT,

which maps the source transport address to the same public address it allocated previously, U. This STUN request arrives at client B (message 19). It generates a response (message 20), which is forwarded to client A (message 21). Based on this check, client A learns that P3 is also reachable from P1. Client B did not learn a new candidate transport address, since U was already known. Now, client A can send media to P3 from either P1 or P2.

9. Mapping ICE into SIP

In this section, we show how to map ICE into SIP. This mapping involves three parts. The first is the actual mapping of the ICE message into SIP and SDP messages, which requires extensions to SDP documented here. The second are security considerations specific to SIP. The third is handling of updates in the offer/answer model.

9.1 Message Mapping

A new SDP attribute is defined to support ICE. It is called "candidate". The candidate attribute MUST be present within a media block of the SDP. It contains a candidate IP address and port (or pair of IP addresses and ports in the case of RTP) that the recipient of the SDP can use. There MAY be multiple candidate 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:

```
candidate-attribute    = "candidate" ":" id SP qvalue SP
                        rtp-user-frag SP rtp-password SP
                        rtp-unicast-address SP rtp-port [SP rtcp-user-frag
                        SP rtcp-password [SP rtcp-unicast-address SP
                        rtcp-port]]
                        ;qvalue from RFC 3261

rtp-port               = port
rtcp-port              = port
rtp-unicast-address    = unicast-address
rtcp-unicast-address   = unicast-address
                        ;unicast-address, port from RFC 2327

rtp-user-frag          = non-ws-string
rtp-password           = non-ws-string
rtcp-user-frag         = non-ws-string
rtcp-password          = non-ws-string
id                     = token
```

With the addition of the candidate 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.

Each media stream element in an ICE message maps to either one or two media blocks in the SDP. If the ICE message has only an IPv4 default address or an IPv6 default address, but not both, one media block is used. If both defaults are present, two media blocks are used. Each default address maps to the m and c lines in the SDP media block. In particular, the <ip-address> from the <rtp-address> element maps into the SDP c line. The <port> from the <rtp-address> maps into the port in the SDP m line. If the ICE message indicates a default RTCP address whose IP address is not identical to the default RTP address, and whose port is not one higher than that of the RTP, the SDP RTCP attribute [2] MUST be used to convey the RTCP transport address.

Each <candidate> element in an ICE message maps to a candidate attribute in the SDP. If the IP version of the <candidate> is IPv4, it MUST be mapped into the media block containing the default IPv4 address. If the IP version of the <candidate> is IPv6, it MUST be mapped into the media block containing the default IPv6 address. Mapping of each individual candidate is simple. The <username-fragment> element of the <rtp-stun-info> element maps to the rtp-user-frag component of the candidate attribute. The <password> element of the <rtp-stun-info> element maps to the rtp-password component of the candidate attribute. The <rtp-address> element maps to the first unicast-address and port components of the candidate attribute.

If the <rtcp-stun-info> element is present, it means that RTCP is in use. The rtcp-user-frag and rtcp-password components of the candidate attribute MUST be present, and MUST be set to the <username-fragment> and <password> elements of the <rtcp-stun-info> element, respectively. If the <rtcp-address> element is also present, its IP address and port information is copied into the rtcp-unicast-address and rtcp-port components of the candidate attribute.

The preference attribute from the <candidate> element is mapped to the q-value component of the candidate attribute. The id attribute from the <candidate> element is mapped into the id component of the candidate attribute.

If the mapping process produced both an IPv6 media block (that is, a media block with an IPv6 address in the c line, and with all IPv6 addresses in the candidate attributes within that block) and an IPv4 media block, these two blocks MUST be grouped using the ANAT grouping [7].

[9.2](#) SIP and SDP Specific Security Considerations

The SDP messages described here contain usernames and passwords. If those passwords are transmitted in the clear, it introduces significant security vulnerabilities, discussed in detail below. In summary, those vulnerabilities would allow an eavesdropper that can inject packets, to "steal" the media streams for a call unless secure media transport (such as SRTP) is used. Even if SRTP is used, an attacker could disrupt a call and prevent media from flowing. These attacks, fortunately, can be obviated by providing secure transport of the SDP. SIP-based implementations of ICE SHOULD use the sips URI scheme when transporting SDP with ICE information, and MAY use S/MIME [\[3\]](#).

[9.3](#) Updates in the Offer/Answer Model

ICE itself only considers an initial exchange of messages. However, the offer/answer model [\[4\]](#) allows for the session to be modified with subsequent exchanges. How is an updated offer with SDP alternate attributes to be treated?

If a user agent receives an updated offer with candidate attributes, it checks to see if it already knows about those candidates. This is done by comparing the transport address and username fragment with existing values. If the combination is already known, no additional action is taken. In particular, if STUN connectivity checks had already been made, no new ones are performed. However, if a candidate contains a new transport address or new username fragment, it is treated as a totally new candidate, and STUN connectivity checks are performed per [Section 5.3.4](#). If a candidate formerly sent by the peer no longer appears, that candidate is considered BAD, and if it was in use previously, it ceases being used, and the next highest priority connection in the GOOD state is used.

The inclusion of the username fragment in the determination of whether a candidate is known provides a hook that allows a peer to request a new set of connectivity checks on an existing transport address. It can update the username fragment and generate an updated offer, without changing the transport address.

[10](#). Security Considerations

STUN itself introduces many security considerations. In particular, there are attacks whereby an eavesdropper replays STUN packets with a modified source address. These modified packets can cause service disruptions and denial-of-service attacks, which are only partially mitigated by the heuristics described in STUN [\[1\]](#).

Interestingly, when STUN is used within ICE, these security weaknesses are mitigated completely, without the need for the heuristics defined in [RFC 3489](#).

Consider an attacker that intercepts a STUN packet used for connectivity checks, and replays it using a faked source address. If successful, this would fool an endpoint into thinking that this faked source address was a valid destination for media (recall that the source transport address of received STUN packets is used as a potential candidate address). However, the recipient of the replayed packet will not just send media to that candidate. It will verify it with a STUN connectivity check. This check will be sent to that faked source address, and if there is no response, the address will not be used. The attacker cannot answer the STUN request without access to the username and password, which are exchanged as part of the signaling. Thus, if the signaling is protected as recommended above, the attacker cannot obtain the username or password.

If an attacker instead intercepts and replays STUN packets used for the purposes of unilateral allocation, a similar result occurs. The target of the attack will be fooled into thinking it has a STUN derived transport address that it does not. Its peer will perform a connectivity check to this address, which will fail. The attacker cannot force this check to succeed without access to the username and password, which are protected. Thus, this address will not be used.

In the worst case, an attacker can generate enough traffic so that none of the valid STUN checks or unilateral allocations succeed. This would result in a service disruption. However, this attack is no worse than any pure packet flood disruption attack launched against any other protocol. These attacks cannot be prevented by any protocol means.

If an attacker could intercept and modify the contents of the Initiate or Accept messages, they could disrupt the session, divert the media, and otherwise take control over the session. This attack is prevented by encryption, authentication and message integrity of the signaling channel used for ICE.

[11.](#) IANA Considerations

[11.1](#) SDP Attribute Name

This specification defines one new SDP attribute per the procedures of [Appendix B of RFC 2327](#). The required information for the registration is:

Contact Name: Jonathan Rosenberg, jdrosen@jdrosen.net.

Attribute Name: candidate

Long Form: candidate

Type of Attribute: media level

Charset Considerations: The attribute is not subject to the charset attribute.

Purpose: This attribute is used with Interactive Connectivity Establishment (ICE), and provides one of many possible candidate addresses for communication. These addresses are validated with an end-to-end connectivity check using Simple Traversal of UDP with NAT (STUN).

Appropriate Values: See [Section 9](#) of RFC XXXX [Note to RFC-ed: please replace XXXX with the RFC number of this specification].

[11.2](#) URN Sub-Namespace Registration

This section registers a new XML namespace, per the guidelines in [\[6\]](#)

URI: The URI for this namespace is `urn:ietf:params:xml:ns:ice`.

Registrant Contact: IETF, MMUSIC working group, (mmusic@ietf.org), Jonathan Rosenberg (jdrosen@jdrosen.net).

XML:

```
BEGIN
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
    "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta http-equiv="content-type"
    content="text/html; charset=iso-8859-1"/>
  <title>ICE Namespace</title>
</head>
<body>
  <h1>Namespace for ICE Documents</h1>
  <h2>urn:ietf:params:xml:ns:ice</h2>
  <p>See <a href="[URL of published
RFC]">RFCXXXX</a>. [Note to RFC-ed: please replace XXXX with the RFC
number of this specification.]</p>
</body>
</html>
END
```

[11.3](#) XML Schema Registration

This section registers an XML schema per the procedures in [\[6\]](#).

URI: urn:ietf:params:xml:schema:ice

Registrant Contact: IETF, MMUSIC working group, (mmusic@ietf.org),
Jonathan Rosenberg (jdrosen@jdrosen.net).

The XML for this schema can be found as the sole content of
[Section 7](#).

[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 [\[14\]](#). 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 significant 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 can be 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 removed 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 network with both 6to4 and native v6 connectivity to determine which address to use when communicating with a peer.

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. Fortunately, those security considerations are also mitigated by ICE.

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 NATP 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, Francois Audet and Magnus Westerland 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] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", [RFC 3264](#), June 2002.
- [5] Zopf, R., "Real-time Transport Protocol (RTP) Payload for Comfort Noise (CN)", [RFC 3389](#), September 2002.
- [6] Mealling, M., "The IETF XML Registry", [BCP 81](#), [RFC 3688](#), January

2004.

- [7] 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.
- [8] Rosenberg, J., "Traversal Using Relay NAT (TURN)", [draft-rosenberg-midcom-turn-05](#) (work in progress), July 2004.

14.2 Informative References

- [9] Schulzrinne, H., Rao, A. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", [RFC 2326](#), April 1998.
- [10] Senie, D., "Network Address Translator (NAT)-Friendly Application Design Guidelines", [RFC 3235](#), January 2002.
- [11] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A. and A. Rayhan, "Middlebox communication architecture and framework", [RFC 3303](#), August 2002.
- [12] Borella, M., Lo, J., Grabelsky, D. and G. Montenegro, "Realm Specific IP: Framework", [RFC 3102](#), October 2001.
- [13] Borella, M., Grabelsky, D., Lo, J. and K. Taniguchi, "Realm Specific IP: Protocol Specification", [RFC 3103](#), October 2001.
- [14] Daigle, L. and IAB, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation", [RFC 3424](#), November 2002.
- [15] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", [RFC 3550](#), July 2003.
- [16] Baugher, M., McGrew, D., Naslund, M., Carrara, E. and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", [RFC 3711](#), March 2004.
- [17] Carpenter, B. and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", [RFC 3056](#), February 2001.
- [18] Huitema, C., "Teredo: Tunneling IPv6 over UDP through NATs", [draft-huitema-v6ops-teredo-02](#) (work in progress), June 2004.
- [19] Hellstrom, G., "RTP Payload for Text Conversation", [draft-ietf-avt-rfc2793bis-09](#) (work in progress), August 2004.

Author's Address

Jonathan Rosenberg
Cisco Systems
600 Lanidex Plaza
Parsippany, NJ 07054
US

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

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in [BCP 78](#) and [BCP 79](#).

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at <http://www.ietf.org/ipr>.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2004). This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

Acknowledgment

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