ICE
Internet-Draft
Obsoletes: 5245 (if approved)
Intended status: Standards Track
Expires: December 15, 2016

A. Keranen
C. Holmberg
Ericsson
J. Rosenberg
jdrosen.net

June 13, 2016

Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal draft-ietf-ice-rfc5245bis-02

Abstract

This document describes a protocol for Network Address Translator (NAT) traversal for UDP-based multimedia. This protocol is called Interactive Connectivity Establishment (ICE). ICE makes use of the Session Traversal Utilities for NAT (STUN) protocol and its extension, Traversal Using Relay NAT (TURN).

This document obsoletes RFC 5245.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on December 15, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction 5
2. Overview of ICE 6
2.1. Gathering Candidate Addresses 8
2.2. Connectivity Checks 10
2.3. Sorting Candidates 11
2.4. Frozen Candidates 12
2.5. Security for Checks 13
2.6. Concluding ICE 13
2.7. Lite Implementations 14
2.8. Usages of ICE 15
3. Terminology 15
4. ICE Candidate Gathering and Exchange 18
4.1. Procedures for Full Implementation 19
4.1.1. Gathering Candidates 19
4.1.1.1. Host Candidates 20
4.1.1.2. Server Reflexive and Relayed Candidates 21
4.1.1.3. Computing Foundations 23
4.1.1.4. Keeping Candidates Alive 23
4.1.2. Prioritizing Candidates 23
4.1.2.1. Recommended Formula 24
4.1.2.2. Guidelines for Choosing Type and Local Preferences 25
4.1.3. Eliminating Redundant Candidates 26
4.2. Lite Implementation Procedures 26
4.3. Encoding the Candidate Information 27
5. ICE Candidate Processing 29
5.1. Procedures for Full Implementation 29
5.1.1. Verifying ICE Support 29

5.1.2. Determining Role 30
5.1.3. Forming the Check Lists 31
5.1.3.1. Forming Candidate Pairs 31
5.1.3.2. Computing Pair Priority and Ordering Pairs . . . 34
5.1.3.3. Pruning the Pairs 34
5.1.3.4. Computing States 34
5.1.4. Scheduling Checks 37
5.2. Lite Implementation Procedures 39
6. Performing Connectivity Checks 39
6.1. STUN Client Procedures 39
6.1.1. Creating Permissions for Relayed Candidates 39
6.1.2. Sending the Request 40
6.1.2.1. PRIORITY 40
6.1.2.2. USE-CANDIDATE 40
6.1.2.3. ICE-CONTROLLED and ICE-CONTROLLING 40
6.1.2.4. Forming Credentials 41
6.1.2.5. DiffServ Treatment 41
6.1.3. Processing the Response 41
6.1.3.1. Failure Cases 41
6.1.3.2. Success Cases 42
6.1.3.2.1. Discovering Peer Reflexive Candidates 42
6.1.3.2.2. Constructing a Valid Pair 43
6.1.3.2.3. Updating Pair States 44
6.1.3.2.4. Updating the Nominated Flag 45
6.1.3.3. Check List and Timer State Updates 45
6.2. STUN Server Procedures 46
6.2.1. Additional Procedures for Full Implementations . . . 46
6.2.1.1. Detecting and Repairing Role Conflicts 47
6.2.1.2. Computing Mapped Address 48
6.2.1.3. Learning Peer Reflexive Candidates 48
6.2.1.4. Triggered Checks 49
6.2.1.5. Updating the Nominated Flag 50
6.2.2. Additional Procedures for Lite Implementations . . . 50
7. Concluding ICE Processing 50
7.1. Procedures for Full Implementations 51
7.1.1. Nominating Pairs 51
7.1.2. Updating States 52
7.2. Procedures for Lite Implementations 53
7.2.1. Peer Is Full 53
7.2.2. Peer Is Lite 53
7.3. Freeing Candidates 54
7.3.1. Full Implementation Procedures 54
7.3.2. Lite Implementation Procedures 54
8. ICE Restarts 55
9. ICE Option 55
10. Keepalives 56
11. Media Handling 57
11.1. Sending Media 57

11.1.1. Procedures for Full Implementations 57
11.1.2. Procedures for Lite Implementations 57
11.1.3. Procedures for All Implementations 58
11.2. Receiving Media 58
12. Extensibility Considerations 58
13. Setting Ta and RTO 59
13.1. Real-time Media Streams 60
13.2. Non-real-time Sessions 61
14. Example 62
15. Security Considerations 67
15.1. Attacks on Connectivity Checks 67
15.2. Attacks on Server Reflexive Address Gathering 69
15.3. Attacks on Relayed Candidate Gathering 70
15.4. Insider Attacks 70
15.4.1. STUN Amplification Attack 71
16. STUN Extensions 71
16.1. New Attributes 71
16.2. New Error Response Codes 72
17. Operational Considerations 72
17.1. NAT and Firewall Types 72
17.2. Bandwidth Requirements 73
17.2.1. STUN and TURN Server Capacity Planning 73
17.2.2. Gathering and Connectivity Checks 73
17.2.3. Keepalives 74
17.3. ICE and ICE-lite 74
17.4. Troubleshooting and Performance Management 74
17.5. Endpoint Configuration 75
18. IANA Considerations 75
18.1. STUN Attributes 75
18.2. STUN Error Responses 75
18.3. ICE Options 75
19. IAB Considerations 76
19.1. Problem Definition 76
19.2. Exit Strategy 77
19.3. Brittleness Introduced by ICE 77
19.4. Requirements for a Long-Term Solution 78
19.5. Issues with Existing NAPT Boxes 79
20. Changes from RFC 5245 79
21. Acknowledgements 79
22. References 80
22.1. Normative References 80
22.2. Informative References 80
Appendix A. Lite and Full Implementations 84
Appendix B. Design Motivations 85
B.1. Pacing of STUN Transactions 85
B.2. Candidates with Multiple Bases 87
B.3. Purpose of the Related Address and Related Port Attributes 89

B.4. Importance of the STUN Username 89
B.5. The Candidate Pair Priority Formula 91
B.6. Why Are Keepalives Needed? 91
B.7. Why Prefer Peer Reflexive Candidates? 92
B.8. Why Are Binding Indications Used for Keepalives? 92
Authors' Addresses 92

1. Introduction

Protocols establishing multimedia sessions between peers typically involve exchanging IP addresses and ports for the media sources and sinks. However this poses challenges when operated through Network Address Translators (NATs) [RFC3235]. These 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 defined for allowing these protocols to operate through NAT. These include Application Layer Gateways (ALGs), the Middlebox Control Protocol [RFC3303], the original Simple Traversal of UDP Through NAT (STUN) [RFC3489] specification, and Realm Specific IP [RFC3102] [RFC3103] along with session description extensions needed to make them work, such as the Session Description Protocol (SDP) [RFC4566] attribute for the Real Time Control Protocol (RTCP) [RFC3605]. 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 complexity and brittleness into the system. What is needed is a single solution that is flexible enough to work well in all situations.

This specification defines Interactive Connectivity Establishment (ICE) as a technique for NAT traversal for UDP-based media streams (though ICE has been extended to handle other transport protocols, such as TCP [RFC6544]). ICE works by exchanging a multiplicity of IP addresses and ports which are then tested for connectivity by peer-to-peer connectivity checks. The IP addresses and ports are exchanged via mechanisms (for example, including in a offer/answer exchange) and the connectivity checks are performed using Session Traversal Utilities for NAT (STUN) specification [RFC5389]. ICE also makes use of Traversal Using Relays around NAT (TURN) [RFC5766], an extension to STUN. Because ICE exchanges a multiplicity of IP addresses and ports for each media stream, it also allows for address

selection for multihomed and dual-stack hosts, and for this reason it deprecates [RFC4091] and [RFC4092].

2. Overview of ICE

In a typical ICE deployment, we have two endpoints (known as ICE AGENTS) that want to communicate. They are able to communicate indirectly via some signaling protocol (such as SIP), by which they can exchange ICE candidates. Note that ICE is not intended for NAT traversal for the signaling protocol, which is assumed to be provided via another mechanism. At the beginning of the ICE process, the agents are ignorant of their own topologies. In particular, they might or might not be behind a NAT (or multiple tiers of NATs). ICE allows the agents to discover enough information about their topologies to potentially find one or more paths by which they can communicate.

Figure 1 shows a typical environment for ICE deployment. The two endpoints are labelled L and R (for left and right, which helps visualize call flows). Both L and R are behind their own respective NATs though they may not be aware of it. The type of NAT and its properties are also unknown. Agents L and R are capable of engaging in an candidate exchange process, whose purpose is to set up a media session between L and R. Typically, this exchange will occur through a signaling (e.g., SIP) server.

In addition to the agents, a signaling server and NATs, ICE is typically used in concert with STUN or TURN servers in the network. Each agent can have its own STUN or TURN server, or they can be the same.

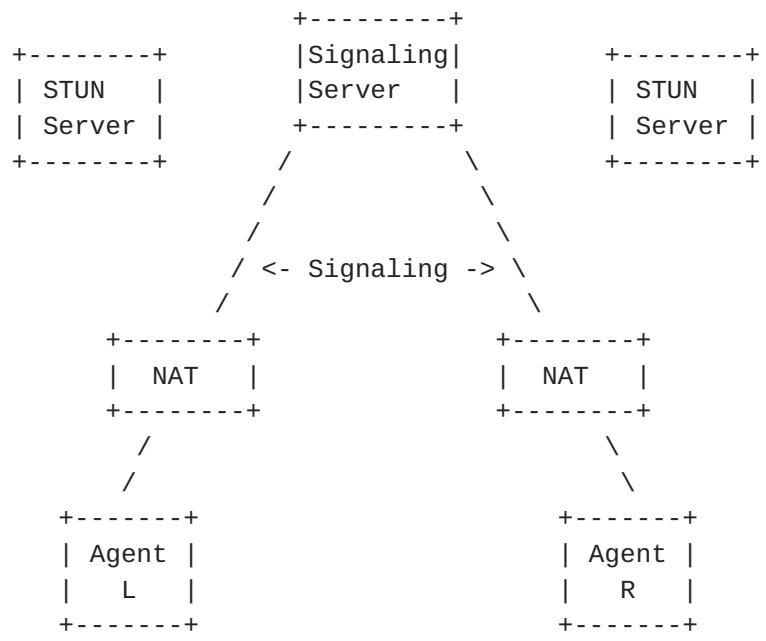


Figure 1: ICE Deployment Scenario

The basic idea behind ICE is as follows: each agent has a variety of candidate TRANSPORT ADDRESSES (combination of IP address and port for a particular transport protocol, which is always UDP in this specification) it could use to communicate with the other agent. These might include:

- A transport address on a directly attached network interface
- A translated transport address on the public side of a NAT (a "server reflexive" address)
- A transport address allocated from a TURN server (a "relayed address")

Potentially, any of L's candidate transport addresses can be used to communicate with any of R's candidate transport addresses. In practice, however, many combinations will not work. For instance, if L and R are both behind NATs, their directly attached interface addresses are unlikely to be able to communicate directly (this is why ICE is needed, after all!). The purpose of ICE is to discover which pairs of addresses will work. The way that ICE does this is to systematically try all possible pairs (in a carefully sorted order) until it finds one or more that work.

2.1. Gathering Candidate Addresses

In order to execute ICE, an agent has to identify all of its address candidates. A CANDIDATE is a transport address -- a combination of IP address and port for a particular transport protocol (with only UDP specified here). This document defines three types of candidates, some derived from physical or logical network interfaces, others discoverable via STUN and TURN. Naturally, one viable candidate is a transport address obtained directly from a local interface. Such a candidate is called a HOST CANDIDATE. The local interface could be Ethernet or WiFi, or it could be one that is obtained through a tunnel mechanism, such as a Virtual Private Network (VPN) or Mobile IP (MIP). In all cases, such a network interface appears to the agent as a local interface from which ports (and thus candidates) can be allocated.

If an agent is multihomed, it obtains a candidate from each IP address. Depending on the location of the PEER (the other agent in the session) on the IP network relative to the agent, the agent may be reachable by the peer through one or more of those IP addresses. Consider, for example, an agent that has a local IP address on a private net 10 network (I1), and a second connected to the public Internet (I2). A candidate from I1 will be directly reachable when communicating with a peer on the same private net 10 network, while a candidate from I2 will be directly reachable when communicating with a peer on the public Internet. Rather than trying to guess which IP address will work, the initiating sends both the candidates to its peer.

Next, the agent uses STUN or TURN to obtain additional candidates. These come in two flavors: translated addresses on the public side of a NAT (SERVER REFLEXIVE CANDIDATES) and addresses on TURN servers (RELAYED CANDIDATES). When TURN servers are utilized, both types of candidates are obtained from the TURN server. If only STUN servers are utilized, only server reflexive candidates are obtained from them. The relationship of these candidates to the host candidate is shown in Figure 2. In this figure, both types of candidates are discovered using TURN. In the figure, the notation X:x means IP address X and UDP port x.

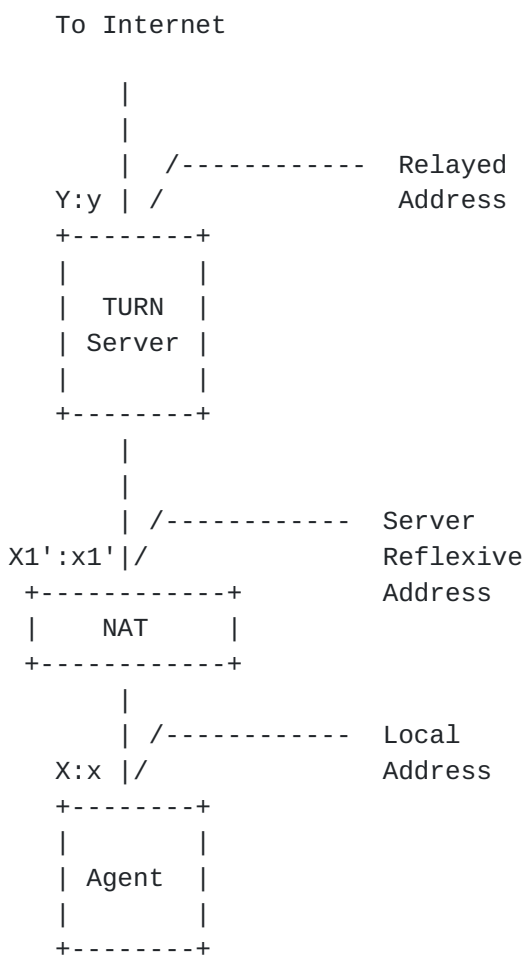


Figure 2: Candidate Relationships

When the agent sends the TURN Allocate request from IP address and port X:x, the NAT (assuming there is one) will create a binding X1':x1', mapping this server reflexive candidate to the host candidate X:x. Outgoing packets sent from the host candidate will be translated by the NAT to the server reflexive candidate. Incoming packets sent to the server reflexive candidate will be translated by the NAT to the host candidate and forwarded to the agent. We call the host candidate associated with a given server reflexive candidate the BASE.

Note: "Base" refers to the address an agent sends from for a particular candidate. Thus, as a degenerate case host candidates also have a base, but it's the same as the host candidate.

When there are multiple NATs between the agent and the TURN server, the TURN request will create a binding on each NAT, but only the outermost server reflexive candidate (the one nearest the TURN

server) will be discovered by the agent. If the agent is not behind a NAT, then the base candidate will be the same as the server reflexive candidate and the server reflexive candidate is redundant and will be eliminated.

The Allocate request then arrives at the TURN server. The TURN server allocates a port y from its local IP address Y, and generates an Allocate response, informing the agent of this relayed candidate. The TURN server also informs the agent of the server reflexive candidate, X1':x1' by copying the source transport address of the Allocate request into the Allocate response. The TURN server acts as a packet relay, forwarding traffic between L and R. In order to send traffic to L, R sends traffic to the TURN server at Y:y, and the TURN server forwards that to X1':x1', which passes through the NAT where it is mapped to X:x and delivered to L.

When only STUN servers are utilized, the agent sends a STUN Binding request [RFC5389] to its STUN server. The STUN server will inform the agent of the server reflexive candidate X1':x1' by copying the source transport address of the Binding request into the Binding response.

2.2. Connectivity Checks

Once L has gathered all of its candidates, it orders them in highest to lowest-priority and sends them to R over the signaling channel. When R receives the candidates from L, it performs the same gathering process and responds with its own list of candidates. At the end of this process, each agent has a complete list of both its candidates and its peer's candidates. It pairs them up, resulting in CANDIDATE PAIRS. To see which pairs work, each agent schedules a series of CHECKS. Each check is a STUN request/response transaction that the client will perform on a particular candidate pair by sending a STUN request from the local candidate to the remote candidate.

The basic principle of the connectivity checks is simple:

1. Sort the candidate pairs in priority order.

2. Send checks on each candidate pair in priority order.

3. Acknowledge checks received from the other agent.

With both agents performing a check on a candidate pair, the result is a 4-way handshake:

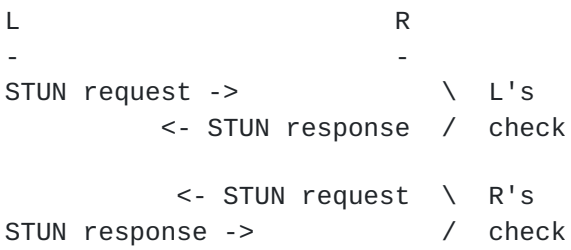


Figure 3: Basic Connectivity Check

It is important to note that the STUN requests are sent to and from the exact same IP addresses and ports that will be used for media (e.g., RTP and RTCP). Consequently, agents demultiplex STUN and RTP/RTCP using contents of the packets, rather than the port on which they are received. Fortunately, this demultiplexing is easy to do, especially for RTP and RTCP.

Because a STUN Binding request is used for the connectivity check, the STUN Binding response will contain the agent's translated transport address on the public side of any NATs between the agent and its peer. If this transport address is different from other candidates the agent already learned, it represents a new candidate, called a PEER REFLEXIVE CANDIDATE, which then gets tested by ICE just the same as any other candidate.

As an optimization, as soon as R gets L's check message, R schedules a connectivity check message to be sent to L on the same candidate pair. This accelerates the process of finding a valid candidate, and is called a TRIGGERED CHECK.

At the end of this handshake, both L and R know that they can send (and receive) messages end-to-end in both directions.

2.3. Sorting Candidates

Because the algorithm above searches all candidate pairs, if a working pair exists it will eventually find it no matter what order the candidates are tried in. In order to produce faster (and better) results, the candidates are sorted in a specified order. The resulting list of sorted candidate pairs is called the CHECK LIST. The algorithm is described in Section 4.1.2 but follows two general principles:

- Each agent gives its candidates a numeric priority, which is sent along with the candidate to the peer.

- The local and remote priorities are combined so that each agent has the same ordering for the candidate pairs.

The second property is important for getting ICE to work when there are NATs in front of L and R. Frequently, NATs will not allow packets in from a host until the agent behind the NAT has sent a packet towards that host. Consequently, ICE checks in each direction will not succeed until both sides have sent a check through their respective NATs.

The agent works through this check list by sending a STUN request for the next candidate pair on the list periodically. These are called ORDINARY CHECKS.

In general, the priority algorithm is designed so that candidates of similar type get similar priorities and so that more direct routes (that is, through fewer media relays and through fewer NATs) are preferred over indirect ones (ones with more media relays and more NATs). Within those guidelines, however, agents have a fair amount of discretion about how to tune their algorithms.

2.4. Frozen Candidates

The previous description only addresses the case where the agents wish to establish a media session with one COMPONENT (a piece of a media stream requiring a single transport address; a media stream may require multiple components, each of which has to work for the media stream as a whole to be work). Sometimes (e.g., with RTP and RTCP in separate components), the agents actually need to establish connectivity for more than one flow.

The network properties are likely to be very similar for each component (especially because RTP and RTCP are sent and received from the same IP address). It is usually possible to leverage information from one media component in order to determine the best candidates for another. ICE does this with a mechanism called "frozen candidates".

Each candidate is associated with a property called its FOUNDATION. Two candidates have the same foundation when they are "similar" -- of the same type and obtained from the same host candidate and STUN/TURN server using the same protocol. Otherwise, their foundation is different. A candidate pair has a foundation too, which is just the concatenation of the foundations of its two candidates. Initially, only the candidate pairs with unique foundations are tested. The other candidate pairs are marked "frozen". When the connectivity checks for a candidate pair succeed, the other candidate pairs with the same foundation are unfrozen. This avoids repeated checking of components that are superficially more attractive but in fact are likely to fail.

While we've described "frozen" here as a separate mechanism for expository purposes, in fact it is an integral part of ICE and the ICE prioritization algorithm automatically ensures that the right candidates are unfrozen and checked in the right order. However, if the ICE usage does not utilize multiple components or media streams, it does not need to implement this algorithm.

2.5. Security for Checks

Because ICE is used to discover which addresses can be used to send media between two agents, it is important to ensure that the process cannot be hijacked to send media to the wrong location. Each STUN connectivity check is covered by a message authentication code (MAC) computed using a key exchanged in the signaling channel. This MAC provides message integrity and data origin authentication, thus stopping an attacker from forging or modifying connectivity check messages. Furthermore, if for example a SIP [RFC3261] caller is using ICE, and their call forks, the ICE exchanges happen independently with each forked recipient. In such a case, the keys exchanged in the signaling help associate each ICE exchange with each forked recipient.

2.6. Concluding ICE

ICE checks are performed in a specific sequence, so that high-priority candidate pairs are checked first, followed by lower-priority ones. One way to conclude ICE is to declare victory as soon as a check for each component of each media stream completes successfully. Indeed, this is a reasonable algorithm, and details for it are provided below. However, it is possible that a packet loss will cause a higher-priority check to take longer to complete. In that case, allowing ICE to run a little longer might produce better results. More fundamentally, however, the prioritization defined by this specification may not yield "optimal" results. As an example, if the aim is to select low-latency media paths, usage of a relay is a hint that latencies may be higher, but it is nothing more than a hint. An actual round-trip time (RTT) measurement could be made, and it might demonstrate that a pair with lower priority is actually better than one with higher priority.

Consequently, ICE assigns one of the agents in the role of the CONTROLLING AGENT, and the other of the CONTROLLED AGENT. The controlling agent gets to nominate which candidate pairs will get used for media amongst the ones that are valid.

When nominating, the controlling agent lets the checks continue until at least one valid candidate pair for each media stream is found. Then, it picks amongst those that are valid, and sends a second STUN

request on its NOMINATED candidate pair, but this time with a flag set to tell the peer that this pair has been nominated for use. This is shown in Figure 4.

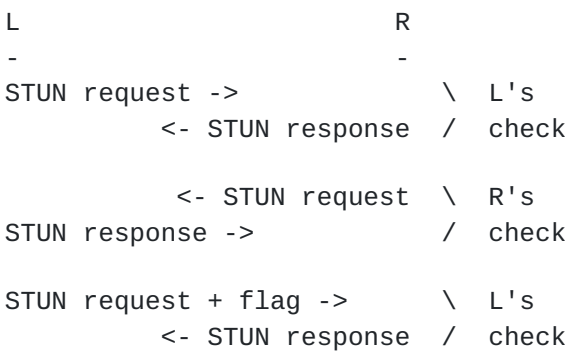


Figure 4: Nomination

Once the STUN transaction with the flag completes, both sides cancel any future checks for that media stream. ICE will now send media using this pair. The pair an ICE agent is using for media is called the SELECTED PAIR.

Once ICE is concluded, it can be restarted at any time for one or all of the media streams by either agent. This is done by sending an updated candidate information indicating a restart.

2.7. Lite Implementations

In order for ICE to be used in a call, both agents need to support it. However, certain agents will always be connected to the public Internet and have a public IP address at which it can receive packets from any correspondent. To make it easier for these devices to support ICE, ICE defines a special type of implementation called LITE (in contrast to the normal FULL implementation). A lite implementation doesn't gather candidates; it includes only host candidates for any media stream. Lite agents do not generate connectivity checks or run the state machines, though they need to be able to respond to connectivity checks. When a lite implementation connects with a full implementation, the full agent takes the role of the controlling agent, and the lite agent takes on the controlled role. When two lite implementations connect, no checks are sent.

For guidance on when a lite implementation is appropriate, see the discussion in Appendix A.

It is important to note that the lite implementation was added to this specification to provide a stepping stone to full implementation. Even for devices that are always connected to the public Internet, a full implementation is preferable if achievable.

2.8. Usages of ICE

This document specifies generic use of ICE with protocols that provide means to exchange candidate information between the ICE Peers. The specific details of (i.e how to encode candidate information and the actual candidate exchange process) for different protocols using ICE are described in separate usage documents. One possible way the agents can exchange the candidate information is to use [RFC3264] based Offer/Answer semantics as part of the SIP [RFC3261] protocol [I-D.ietf-mmusic-ice-sip-sdp].

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Readers should be familiar with the terminology defined in the STUN [RFC5389], and NAT Behavioral requirements for UDP [RFC4787].

This specification makes use of the following additional terminology:

ICE Agent: An agent is the protocol implementation involved in the ICE candidate exchange. There are two agents involved in a typical candidate exchange.

Initiating Peer, Initiating Agent, Initiator: An initiating agent is the protocol implementation involved in the ICE candidate exchange that initiates the ICE candidate exchange process.

Responding Peer, Responding Agent, Responder: A receiving agent is the protocol implementation involved in the ICE candidate exchange that receives and responds to the candidate exchange process initiated by the Initiator.

ICE Candidate Exchange, Candidate Exchange: The process where the ICE agents exchange information (e.g., candidates and passwords) that is needed to perform ICE. [RFC3264] Offer/Answer with SDP encoding is one example of a protocol that can be used for exchanging the candidate information.

Peer: From the perspective of one of the agents in a session, its peer is the other agent. Specifically, from the perspective of the initiating agent, the peer is the responding agent. From the perspective of the responding agent, the peer is the initiating agent.

Transport Address: The combination of an IP address and transport protocol (such as UDP or TCP) port.

Media, Media Stream, Media Session: When ICE is used to setup multimedia sessions, the media is usually transported over RTP, and a media stream composes of a stream of RTP packets. When ICE is used with other than multimedia sessions, the terms "media", "media stream", and "media session" are still used in this specification to refer to the IP data packets that are exchanged between the peers on the path created and tested with ICE.

Candidate, Candidate Information: A transport address that is a potential point of contact for receipt of media. Candidates also have properties -- their type (server reflexive, relayed, or host), priority,foundation, and base.

Component: A component is a piece of a media stream requiring a single transport address; a media stream may require multiple components, each of which has to work for the media stream as a whole to work. For media streams based on RTP, unless RTP and RTCP are multiplexed in the same port, there are two components per media stream -- one for RTP, and one for RTCP.

Host Candidate: A candidate obtained by binding to a specific port from an IP address on the host. This includes IP addresses on physical interfaces and logical ones, such as ones obtained through Virtual Private Networks (VPNs) and Realm Specific IP (RSIP) [RFC3102] (which lives at the operating system level).

Server Reflexive Candidate: A candidate whose IP address and port are a binding allocated by a NAT for an agent when it sent a packet through the NAT to a server. Server reflexive candidates can be learned by STUN servers using the Binding request, or TURN servers, which provides both a relayed and server reflexive candidate.

Peer Reflexive Candidate: A candidate whose IP address and port are a binding allocated by a NAT for an agent when it sent a STUN Binding request through the NAT to its peer.

Relayed Candidate: A candidate obtained by sending a TURN Allocate request from a host candidate to a TURN server. The relayed

candidate is resident on the TURN server, and the TURN server relays packets back towards the agent.

Base: The base of a server reflexive candidate is the host candidate from which it was derived. A host candidate is also said to have a base, equal to that candidate itself. Similarly, the base of a relayed candidate is that candidate itself.

Foundation: An arbitrary string that is the same for two candidates that have the same type, base IP address, protocol (UDP, TCP, etc.), and STUN or TURN server. If any of these are different, then the foundation will be different. Two candidate pairs with the same foundation pairs are likely to have similar network characteristics. Foundations are used in the frozen algorithm.

Local Candidate: A candidate that an agent has obtained and shared with the peer.

Remote Candidate: A candidate that an agent received from its peer.

Default Destination/Candidate: The default destination for a component of a media stream is the transport address that would be used by an agent that is not ICE aware. A default candidate for a component is one whose transport address matches the default destination for that component.

Candidate Pair: A pairing containing a local candidate and a remote candidate.

Check, Connectivity Check, STUN Check: A STUN Binding request transaction for the purposes of verifying connectivity. A check is sent from the local candidate to the remote candidate of a candidate pair.

Check List: An ordered set of candidate pairs that an agent will use to generate checks.

Ordinary Check: A connectivity check generated by an agent as a consequence of a timer that fires periodically, instructing it to send a check.

Triggered Check: A connectivity check generated as a consequence of the receipt of a connectivity check from the peer.

Valid List: An ordered set of candidate pairs for a media stream that have been validated by a successful STUN transaction.

Full: An ICE implementation that performs the complete set of functionality defined by this specification.

Lite: An ICE implementation that omits certain functions, implementing only as much as is necessary for a peer implementation that is full to gain the benefits of ICE. Lite implementations do not maintain any of the state machines and do not generate connectivity checks.

Controlling Agent: The ICE agent that is responsible for selecting the final choice of candidate pairs and signaling them through STUN. In any session, one agent is always controlling. The other is the controlled agent.

Controlled Agent: An ICE agent that waits for the controlling agent to select the final choice of candidate pairs.

Nomination, Regular Nomination: The process of picking a valid candidate pair for media traffic by validating the pair with one STUN request, and then picking it by sending a second STUN request with a flag indicating its nomination.

Nominated: If a valid candidate pair has its nominated flag set, it means that it may be selected by ICE for sending and receiving media.

Selected Pair, Selected Candidate: The candidate pair selected by ICE for sending and receiving media is called the selected pair, and each of its candidates is called the selected candidate.

Using Protocol, ICE Usage: The protocol that uses ICE for NAT traversal. A usage specification defines the protocol specific details on how the procedures defined here are applied to that protocol.

4. ICE Candidate Gathering and Exchange

As part of ICE processing, both the initiating and responding agents exchange encoded candidate information as defined by the Usage Protocol (ICE Usage). Specifics of encoding mechanism and the semantics of candidate information exchange is out of scope of this specification.

However at a higher level, the below diagram captures ICE processing sequence in the agents (initiator and responder) for exchange of their respective candidate(s) information.

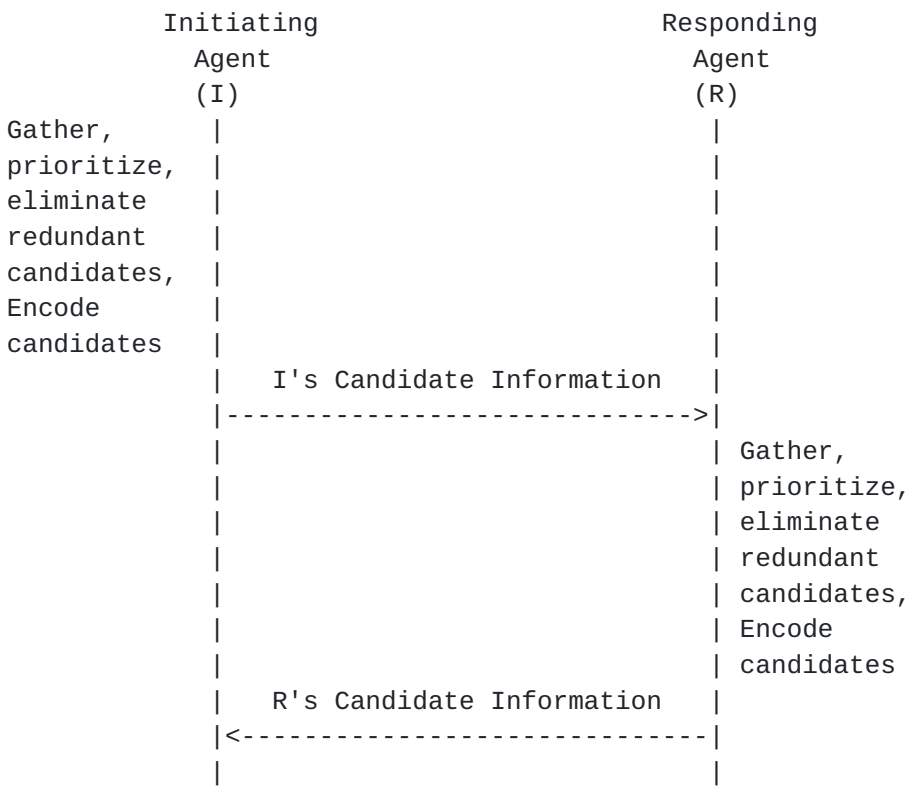


Figure 5: Candidate Gathering and Exchange Sequence

As shown, the agents involved in the candidate exchange perform (1) candidate gathering, (2) candidate prioritization, (3) eliminating redundant candidates, (4) (possibly) choose default candidates, and then (5) formulate and send the candidates to the Peer ICE agent. All but the last of these five steps differ for full and lite implementations.

4.1. Procedures for Full Implementation

4.1.1. Gathering Candidates

An agent gathers candidates when it believes that communication is imminent. An initiating agent can do this based on a user interface cue, or based on an explicit request to initiate a session. Every candidate is a transport address. It also has a type and a base. Four types are defined and gathered by this specification -- host candidates, server reflexive candidates, peer reflexive candidates, and relayed candidates. The server reflexive candidates are gathered using STUN or TURN, and relayed candidates are obtained through TURN. Peer reflexive candidates are obtained in later phases of ICE, as a consequence of connectivity checks. The base of a candidate is the candidate that an agent must send from when using that candidate.

The process for gathering candidates at the responding agent is identical to the process for the initiating agent. It is RECOMMENDED that the responding agent begins this process immediately on receipt of the candidate information, prior to alerting the user. Such gathering MAY begin when an agent starts.

4.1.1.1. Host Candidates

The first step is to gather host candidates. Host candidates are obtained by binding to ports (typically ephemeral) on a IP address attached to an interface (physical or virtual, including VPN interfaces) on the host.

For each UDP media stream the agent wishes to use, the agent SHOULD obtain a candidate for each component of the media stream on each IP address that the host has, with the exceptions listed below. The agent obtains each candidate by binding to a UDP port on the specific IP address. A host candidate (and indeed every candidate) is always associated with a specific component for which it is a candidate.

Each component has an ID assigned to it, called the component ID. For RTP-based media streams, unless both RTP and RTCP are multiplexed in the same UDP port (RTP/RTCP multiplexing), the RTP itself has a component ID of 1, and RTCP a component ID of 2. In case of RTP/RTCP multiplexing, a component ID of 1 is used for both RTP and RTCP.

When candidates are obtained, unless the agent knows for sure that RTP/RTCP multiplexing will be used (i.e. the agent knows that the other agent also supports, and is willing to use, RTP/RTCP multiplexing), or unless the agent only supports RTP/RTCP multiplexing, the agent MUST obtain a separate candidate for RTCP. If an agent has obtained a candidate for RTCP, and ends up using RTP/RTCP multiplexing, the agent does not need to perform connectivity checks on the RTCP candidate.

If an agent is using separate candidates for RTP and RTCP, it will end up with 2*K host candidates if an agent has K IP addresses.

Note that the responding agent, when obtaining its candidates, will typically know if the other agent supports RTP/RTCP multiplexing, in which case it will not need to obtain a separate candidate for RTCP. However, absence of a component ID 2 as such does not imply use of RTCP/RTP multiplexing, as it could also mean that RTCP is not used.

For other than RTP-based streams, use of multiple components is discouraged since using them increases the complexity of ICE processing. If multiple components are needed, the component IDs SHOULD start with 1 and increase by 1 for each component.

The base for each host candidate is set to the candidate itself.

The host candidates are gathered from all IP addresses with the following exceptions:

- Addresses from a loopback interface MUST NOT be included in the candidate addresses.

- Deprecated IPv4-compatible IPv6 addresses [RFC4291] and IPv6 site-local unicast addresses [RFC3879] MUST NOT be included in the address candidates.

- IPv4-mapped IPv6 addresses SHOULD NOT be included in the offered candidates unless the application using ICE does not support IPv4 (i.e., is an IPv6-only application [RFC4038]).

- If one or more host candidates corresponding to an IPv6 address generated using a mechanism that prevents location tracking [I-D.ietf-6man-ipv6-address-generation-privacy] are gathered, host candidates corresponding to IPv6 addresses that do allow location tracking, that are configured on the same interface, and are part of the same network prefix MUST NOT be gathered; and host candidates corresponding to IPv6 link-local addresses MUST NOT be gathered.

4.1.1.2. Server Reflexive and Relayed Candidates

Agents SHOULD obtain relayed candidates and SHOULD obtain server reflexive candidates. These requirements are at SHOULD strength to allow for provider variation. Use of STUN and TURN servers may be unnecessary in closed networks where agents are never connected to the public Internet or to endpoints outside of the closed network. In such cases, a full implementation would be used for agents that are dual-stack or multihomed, to select a host candidate. Use of TURN servers is expensive, and when ICE is being used, they will only be utilized when both endpoints are behind NATs that perform address and port dependent mapping. Consequently, some deployments might consider this use case to be marginal, and elect not to use TURN servers. If an agent does not gather server reflexive or relayed candidates, it is RECOMMENDED that the functionality be implemented and just disabled through configuration, so that it can be re-enabled through configuration if conditions change in the future.

If an agent is gathering both relayed and server reflexive candidates, it uses a TURN server. If it is gathering just server reflexive candidates, it uses a STUN server.

The agent next pairs each host candidate with the STUN or TURN server with which it is configured or has discovered by some means. If a STUN or TURN server is configured, it is RECOMMENDED that a domain name be configured, and the DNS procedures in [RFC5389] (using SRV records with the "stun" service) be used to discover the STUN server, and the DNS procedures in [RFC5766] (using SRV records with the "turn" service) be used to discover the TURN server.

This specification only considers usage of a single STUN or TURN server. When there are multiple choices for that single STUN or TURN server (when, for example, they are learned through DNS records and multiple results are returned), an agent SHOULD use a single STUN or TURN server (based on its IP address) for all candidates for a particular session. This improves the performance of ICE. The result is a set of pairs of host candidates with STUN or TURN servers. The agent then chooses one pair, and sends a Binding or Allocate request to the server from that host candidate. Binding requests to a STUN server are not authenticated, and any ALTERNATE-SERVER attribute in a response is ignored. Agents MUST support the backwards compatibility mode for the Binding request defined in [RFC5389]. Allocate requests SHOULD be authenticated using a long-term credential obtained by the client through some other means.

Every Ta milliseconds thereafter, the agent can generate another new STUN or TURN transaction. This transaction can either be a retry of a previous transaction that failed with a recoverable error (such as authentication failure), or a transaction for a new host candidate and STUN or TURN server pair. The agent SHOULD NOT generate transactions more frequently than one every Ta milliseconds. See Section 13 for guidance on how to set Ta and the STUN retransmit timer, RTO.

The agent will receive a Binding or Allocate response. A successful Allocate response will provide the agent with a server reflexive candidate (obtained from the mapped address) and a relayed candidate in the XOR-RELAYED-ADDRESS attribute. If the Allocate request is rejected because the server lacks resources to fulfill it, the agent SHOULD instead send a Binding request to obtain a server reflexive candidate. A Binding response will provide the agent with only a server reflexive candidate (also obtained from the mapped address). The base of the server reflexive candidate is the host candidate from which the Allocate or Binding request was sent. The base of a relayed candidate is that candidate itself. If a relayed candidate is identical to a host candidate (which can happen in rare cases), the relayed candidate MUST be discarded.

If an IPv6-only agent is in a network that utilizes NAT64 [RFC6146] and DNS64 [RFC6147] technologies, it may gather also IPv4 server

reflexive and/or relayed candidates from IPv4-only STUN or TURN servers. IPv6-only agents SHOULD also utilize IPv6 prefix discovery [RFC7050] to discover the IPv6 prefix used by NAT64 (if any) and generate server reflexive candidates for each IPv6-only interface accordingly. The NAT64 server reflexive candidates are prioritized like IPv4 server reflexive candidates.

4.1.1.3. Computing Foundations

Finally, the agent assigns each candidate a foundation. The foundation is an identifier, scoped within a session. Two candidates MUST have the same foundation ID when all of the following are true:

- they are of the same type (host, relayed, server reflexive, or peer reflexive)
- their bases have the same IP address (the ports can be different)
- for reflexive and relayed candidates, the STUN or TURN servers used to obtain them have the same IP address
- they were obtained using the same transport protocol (TCP, UDP, etc.)

Similarly, two candidates MUST have different foundations if their types are different, their bases have different IP addresses, the STUN or TURN servers used to obtain them have different IP addresses, or their transport protocols are different.

4.1.1.4. Keeping Candidates Alive

Once server reflexive and relayed candidates are allocated, they MUST be kept alive until ICE processing has completed, as described in Section 7.3. For server reflexive candidates learned through a Binding request, the bindings MUST be kept alive by additional Binding requests to the server. Refreshes for allocations are done using the Refresh transaction, as described in [RFC5766]. The Refresh requests will also refresh the server reflexive candidate.

4.1.2. Prioritizing Candidates

The prioritization process results in the assignment of a priority to each candidate. Each candidate for a media stream MUST have a unique priority that MUST be a positive integer between 1 and (2**31 - 1). This priority will be used by ICE to determine the order of the connectivity checks and the relative preference for candidates.

An agent SHOULD compute this priority using the formula in Section 4.1.2.1 and choose its parameters using the guidelines in Section 4.1.2.2. If an agent elects to use a different formula, ICE will take longer to converge since both agents will not be coordinated in their checks.

The process for prioritizing candidates is common across the initiating and the responding agent.

4.1.2.1. Recommended Formula

When using the formula, an agent computes the priority by determining a preference for each type of candidate (server reflexive, peer reflexive, relayed, and host), and, when the agent is multihomed, choosing a preference for its IP addresses. These two preferences are then combined to compute the priority for a candidate. That priority is computed using the following formula:

```
priority = (2^24)*(type preference) +
           (2^8)*(local preference) +
           (2^0)*(256 - component ID)
```

The type preference MUST be an integer from 0 to 126 inclusive, and represents the preference for the type of the candidate (where the types are local, server reflexive, peer reflexive, and relayed). A 126 is the highest preference, and a 0 is the lowest. Setting the value to a 0 means that candidates of this type will only be used as a last resort. The type preference MUST be identical for all candidates of the same type and MUST be different for candidates of different types. The type preference for peer reflexive candidates MUST be higher than that of server reflexive candidates. Note that candidates gathered based on the procedures of Section 4.1.1 will never be peer reflexive candidates; candidates of these type are learned from the connectivity checks performed by ICE.

The local preference MUST be an integer from 0 to 65535 inclusive. It represents a preference for the particular IP address from which the candidate was obtained. 65535 represents the highest preference, and a zero, the lowest. When there is only a single IP address, this value SHOULD be set to 65535. More generally, if there are multiple candidates for a particular component for a particular media stream that have the same type, the local preference MUST be unique for each one. In this specification, this only happens for multihomed hosts or if an agent is using multiple TURN servers. If a host is multihomed because it is dual-stack, the local preference SHOULD be set equal to the precedence value for IP addresses described in RFC

6724 [RFC6724]. If the host operating system provides an API for discovering preference among different addresses, those preferences SHOULD be used for the local preference to prioritize addresses indicated as preferred by the operating system.

The component ID is the component ID for the candidate, and MUST be between 1 and 256 inclusive.

4.1.2.2. Guidelines for Choosing Type and Local Preferences

One criterion for selection of the type and local preference values is the use of a media intermediary, such as a TURN server, VPN server, or NAT. With a media intermediary, if media is sent to that candidate, it will first transit the media intermediary before being received. Relayed candidates are one type of candidate that involves a media intermediary. Another are host candidates obtained from a VPN interface. When media is transited through a media intermediary, 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 media intermediary run by a provider. If these concerns are important, the type preference for relayed candidates SHOULD be lower than host candidates. The RECOMMENDED values are 126 for host candidates, 100 for server reflexive candidates, 110 for peer reflexive candidates, and 0 for relayed candidates.

Furthermore, if an agent is multihomed and has multiple IP addresses, the local preference for host candidates from a VPN interface SHOULD have a priority of 0. If multiple TURN servers are used, local priorities for the candidates obtained from the TURN servers are chosen in a similar fashion as for multihomed local candidates: the local preference value is used to indicate preference among different servers but the preference MUST be unique for each one.

Another criterion for selection of preferences 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) [RFC3056]. It can also help with hosts that have both a native IPv6 address and a 6to4 address. In such a case, higher local preferences could be assigned to the v6 addresses, followed by the 6to4 addresses, followed by the v4 addresses. This allows a site to obtain and begin using native v6 addresses immediately, yet still fall back to 6to4 addresses when communicating with agents in other sites that do not yet have native v6 connectivity.

Another criterion for selecting preferences is security. If a user is a telecommuter, and therefore connected to a corporate network and a local home network, the user 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. In such a case, a VPN address would have a higher local preference than any other address.

Another criterion for selecting preferences is topological awareness. This is most useful for candidates that make use of intermediaries. In those cases, if an agent has preconfigured or dynamically discovered knowledge of the topological proximity of the intermediaries to itself, it can use that to assign higher local preferences to candidates obtained from closer intermediaries.

4.1.3. Eliminating Redundant Candidates

Next, the agent eliminates redundant candidates. A candidate is redundant if its transport address equals another candidate, and its base equals the base of that other candidate. Note that two candidates can have the same transport address yet have different bases, and these would not be considered redundant. Frequently, a server reflexive candidate and a host candidate will be redundant when the agent is not behind a NAT. The agent SHOULD eliminate the redundant candidate with the lower priority.

This process is common across the initiating and responding agents.

4.2. Lite Implementation Procedures

Lite implementations only utilize host candidates. A lite implementation MUST, for each component of each media stream, allocate zero or one IPv4 candidates. It MAY allocate zero or more IPv6 candidates, but no more than one per each IPv6 address utilized by the host. Since there can be no more than one IPv4 candidate per component of each media stream, if an agent has multiple IPv4 addresses, it MUST choose one for allocating the candidate. If a host is dual-stack, it is RECOMMENDED that it allocate one IPv4 candidate and one global IPv6 address. With the lite implementation, ICE cannot be used to dynamically choose amongst candidates. Therefore, including more than one candidate from a particular scope is NOT RECOMMENDED, since only a connectivity check can truly determine whether to use one address or the other.

Each component has an ID assigned to it, called the component ID. For RTP-based media streams, unless RTCP is multiplexed in the same port with RTP, the RTP itself has a component ID of 1, and RTCP a

component ID of 2. If an agent is using RTCP without multiplexing, it MUST obtain candidates for it. However, absence of a component ID 2 as such does not imply use of RTCP/RTP multiplexing, as it could also mean that RTCP is not used.

Each candidate is assigned a foundation. The foundation MUST be different for two candidates allocated from different IP addresses, and MUST be the same otherwise. A simple integer that increments for each IP address will suffice. In addition, each candidate MUST be assigned a unique priority amongst all candidates for the same media stream. This priority SHOULD be equal to:

```
priority = (2^24)*(126) +
           (2^8)*(IP precedence) +
           (2^0)*(256 - component ID)
```

If a host is v4-only, it SHOULD set the IP precedence to 65535. If a host is v6 or dual-stack, the IP precedence SHOULD be the precedence value for IP addresses described in RFC 6724 [RFC6724].

Next, an agent chooses a default candidate for each component of each media stream. If a host is IPv4-only, there would only be one candidate for each component of each media stream, and therefore that candidate is the default. If a host is IPv6 or dual-stack, the selection of default is a matter of local policy. This default SHOULD be chosen such that it is the candidate most likely to be used with a peer. For IPv6-only hosts, this would typically be a globally scoped IPv6 address. For dual-stack hosts, the IPv4 address is RECOMMENDED.

The procedures in this section is common across the initiating and responding agents.

4.3. Encoding the Candidate Information

Regardless of the agent being an Initiator or Responder Agent, the following parameters and their data types needs to be conveyed as part of the candidate exchange process. The specifics of syntax for encoding the candidate information is out of scope of this specification.

Candidate attribute There will be one or more of these for each "media stream". Each candidate is composed of:

Connection Address: The IP address and transport protocol port of the candidate.

Transport: An indicator of the transport protocol for this candidate. This need not be present if the using protocol will only ever run over a single transport protocol. If it runs over more than one, or if others are anticipated to be used in the future, this should be present.

Foundation: A sequence of up to 32 characters.

Component-ID: This would be present only if the using protocol were utilizing the concept of components. If it is, it would be a positive integer that indicates the component ID for which this is a candidate.

Priority: An encoding of the 32-bit priority value.

Candidate Type: The candidate type, as defined in ICE.

Related Address and Port: The related IP address and port for this candidate, as defined by ICE. These MAY be omitted or set to invalid values if the agent does not want to reveal them, e.g., for privacy reasons.

Extensibility Parameters: The using protocol should define some means for adding new per-candidate ICE parameters in the future.

Lite Flag: If ICE lite is used by the using protocol, it needs to convey a boolean parameter which indicates whether the implementation is lite or not.

Connectivity check pacing value: If an agent wants to use other than the default pacing values for the connectivity checks, it MUST indicate this in the ICE exchange.

Username Fragment and Password: The using protocol has to convey a username fragment and password. The username fragment MUST contain at least 24 bits of randomness, and the password MUST contain at least 128 bits of randomness.

ICE extensions: In addition to the per-candidate extensions above, the using protocol should allow for new media-stream or session-level attributes (ice-options).

If the using protocol is using the ICE mismatch feature, a way is needed to convey this parameter in answers. It is a boolean flag.

The exchange of parameters is symmetric; both agents need to send the same set of attributes as defined above.

The using protocol may (or may not) need to deal with backwards compatibility with older implementations that do not support ICE. If the fallback mechanism is being used, then presumably the using protocol provides a way of conveying the default candidate (its IP address and port) in addition to the ICE parameters.

STUN connectivity checks between agents are authenticated using the short-term credential mechanism defined for STUN [RFC5389]. This mechanism relies on a username and password that are exchanged through protocol machinery between the client and server. The username part of this credential is formed by concatenating a username fragment from each agent, separated by a colon. Each agent also provides a password, used to compute the message integrity for requests it receives. The username fragment and password are exchanged between the peers. In addition to providing security, the username provides disambiguation and correlation of checks to media streams. See Appendix B.4 for motivation.

If the initiating agent is a lite implementation, it MUST indicate this when sending its candidates .

ICE provides for extensibility by allowing an agent to include a series of tokens that identify ICE extensions as part of the candidate exchange process.

Once an agent has sent its candidate information, that agent MUST be prepared to receive both STUN and media packets on each candidate. As discussed in Section 11.1, media packets can be sent to a candidate prior to its appearance as the default destination for media.

5. ICE Candidate Processing

Once an agent has candidates from it's peer, it will check if the peer supports ICE, determine its own role, exchanges candidates (Section 4) and for full implementations, forms the check lists and begins connectivity checks as explained in this section.

5.1. Procedures for Full Implementation

5.1.1. Verifying ICE Support

Certain middleboxes, such as ALGs, may alter the ICE candidate information that breaks ICE. If the using protocol is vulnerable to this kind of changes, called ICE mismatch, the responding agent needs to detect this and signal this back to the initiating agent. The details on whether this is needed and how it is done is defined by

the usage specifications. One exception to the above is that an initiating agent would never indicate ICE mismatch.

5.1.2. Determining Role

For each session, each agent (Initiating and Responding) takes on a role. There are two roles -- controlling and controlled. The controlling agent is responsible for the choice of the final candidate pairs used for communications. For a full agent, this means nominating the candidate pairs that can be used by ICE for each media stream, and for updating the peer with the ICE's selection, when needed. The controlled agent is told which candidate pairs to use for each media stream, and does not require updating the peer to signal this information. The sections below describe in detail the actual procedures followed by controlling and controlled nodes.

The rules for determining the role and the impact on behavior are as follows:

Both agents are full: The Initiating Agent which started the ICE processing MUST take the controlling role, and the other MUST take the controlled role. Both agents will form check lists, run the ICE state machines, and generate connectivity checks. The controlling agent will execute the logic in Section 7.1 to nominate pairs that will be selected by ICE, and then both agents end ICE as described in Section 7.1.2.

One agent full, one lite: The full agent MUST take the controlling role, and the lite agent MUST take the controlled role. The full agent will form check lists, run the ICE state machines, and generate connectivity checks. That agent will execute the logic in Section 7.1 to nominate pairs that will be selected by ICE, and use the logic in Section 7.1.2 to end ICE. The lite implementation will just listen for connectivity checks, receive them and respond to them, and then conclude ICE as described in Section 7.2. For the lite implementation, the state of ICE processing for each media stream is considered to be Running, and the state of ICE overall is Running.

Both lite: The Initiating Agent which started the ICE processing MUST take the controlling role, and the other MUST take the controlled role. In this case, no connectivity checks are ever sent. Rather, once the candidates are exchanged, each agent performs the processing described in Section 7 without connectivity checks. It is possible that both agents will believe they are controlled or controlling. In the latter case, the conflict is resolved through glare detection capabilities in the signaling protocol enabling the candidate exchange. The state of

ICE processing for each media stream is considered to be Running, and the state of ICE overall is Running.

Once roles are determined for a session, they persist unless ICE is restarted. An ICE restart causes a new selection of roles and tie-breakers.

5.1.3. Forming the Check Lists

There is one check list per in-use media stream resulting from the candidate exchange. To form the check list for a media stream, the agent forms candidate pairs, computes a candidate pair priority, orders the pairs by priority, prunes them, and sets their states. These steps are described in this section.

5.1.3.1. Forming Candidate Pairs

First, the agent takes each of its candidates for a media stream (called LOCAL CANDIDATES) and pairs them with the candidates it received from its peer (called REMOTE CANDIDATES) for that media stream. In order to prevent the attacks described in Section 15.4.1, agents MAY limit the number of candidates they'll accept in an candidate exchange process. A local candidate is paired with a remote candidate if and only if the two candidates have the same component ID and have the same IP address version. It is possible that some of the local candidates won't get paired with remote candidates, and some of the remote candidates won't get paired with local candidates. This can happen if one agent doesn't include candidates for the all of the components for a media stream. If this happens, the number of components for that media stream is effectively reduced, and considered to be equal to the minimum across both agents of the maximum component ID provided by each agent across all components for the media stream.

In the case of RTP, this would happen when one agent provides candidates for RTCP, and the other does not. As another example, the initiating agent can multiplex RTP and RTCP on the same port [RFC5761]. However, since the initiating agent doesn't know if the peer agent can perform such multiplexing, it includes candidates for RTP and RTCP on separate ports. If the peer agent can perform such multiplexing, it would include just a single component for each candidate -- for the combined RTP/RTCP mux. ICE would end up acting as if there was just a single component for this candidate.

With IPv6 it is common for a host to have multiple host candidates for each interface. To keep the amount of resulting candidate pairs reasonable and to avoid candidate pairs that are highly unlikely to

work, IPv6 link-local addresses [RFC4291] MUST NOT be paired with other than link-local addresses.

The candidate pairs whose local and remote candidates are both the default candidates for a particular component is called, unsurprisingly, the default candidate pair for that component. This is the pair that would be used to transmit media if both agents had not been ICE aware.

In order to aid understanding, Figure 6 shows the relationships between several key concepts -- transport addresses, candidates, candidate pairs, and check lists, in addition to indicating the main properties of candidates and candidate pairs.

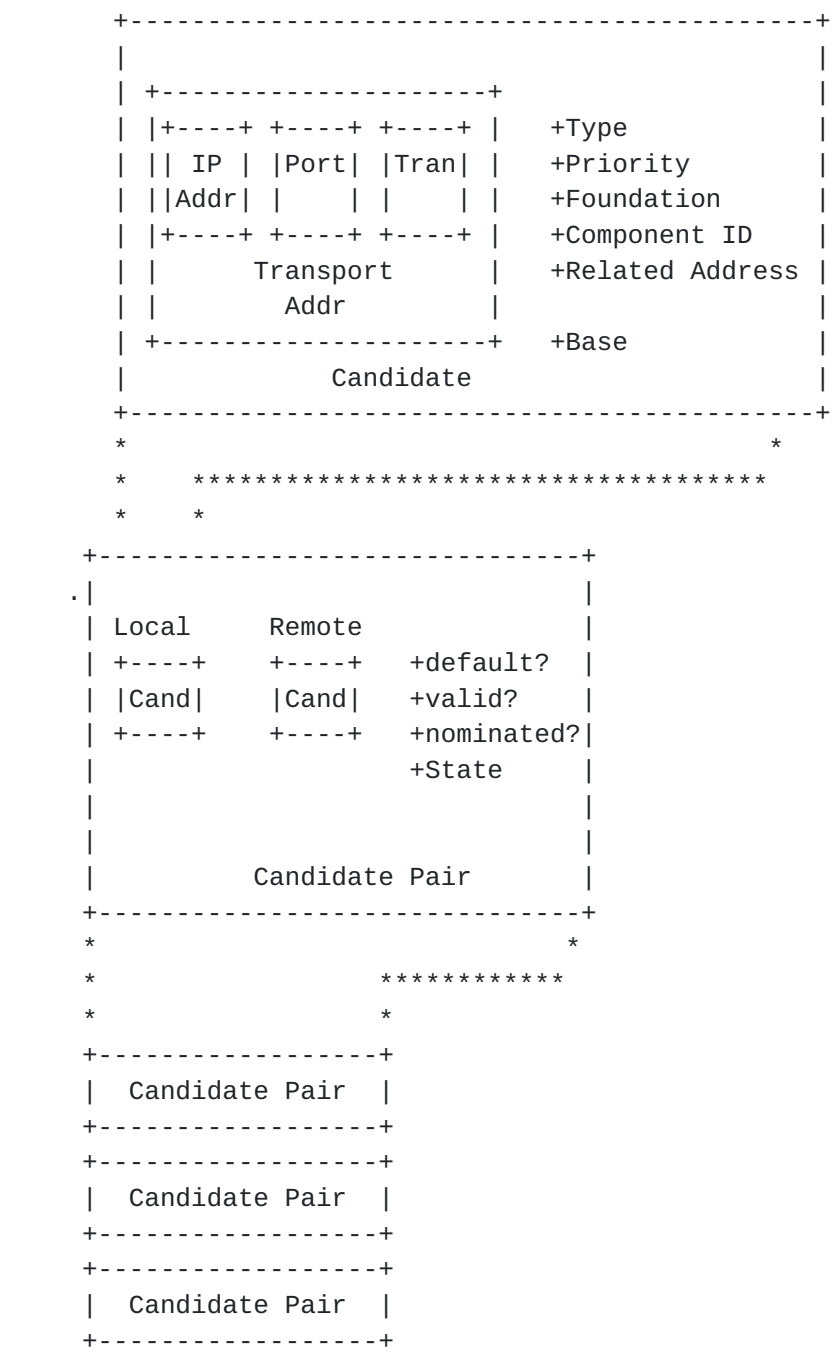


Figure 6: Conceptual Diagram of a Check List

5.1.3.2. Computing Pair Priority and Ordering Pairs

Once the pairs are formed, a candidate pair priority is computed. Let G be the priority for the candidate provided by the controlling agent. Let D be the priority for the candidate provided by the controlled agent. The priority for a pair is computed as:

```
pair priority = 2^32*MIN(G,D) + 2*MAX(G,D) + (G>D?1:0)
```

Where G>D?1:0 is an expression whose value is 1 if G is greater than D, and 0 otherwise. Once the priority is assigned, the agent sorts the candidate pairs in decreasing order of priority. If two pairs have identical priority, the ordering amongst them is arbitrary.

5.1.3.3. Pruning the Pairs

This sorted list of candidate pairs is used to determine a sequence of connectivity checks that will be performed. Each check involves sending a request from a local candidate to a remote candidate. Since an agent cannot send requests directly from a reflexive candidate, but only from its base, the agent next goes through the sorted list of candidate pairs. For each pair where the local candidate is server reflexive, the server reflexive candidate MUST be replaced by its base. Once this has been done, the agent MUST prune the list. This is done by removing a pair if its local and remote candidates are identical to the local and remote candidates of a pair higher up on the priority list. The result is a sequence of ordered candidate pairs, called the check list for that media stream.

In addition, in order to limit the attacks described in Section 15.4.1, an agent MUST limit the total number of connectivity checks the agent performs across all check lists to a specific value, and this value MUST be configurable. A default of 100 is RECOMMENDED. This limit is enforced by discarding the lower-priority candidate pairs until there are less than 100. It is RECOMMENDED that a lower value be utilized when possible, set to the maximum number of plausible checks that might be seen in an actual deployment configuration. The requirement for configuration is meant to provide a tool for fixing this value in the field if, once deployed, it is found to be problematic.

5.1.3.4. Computing States

Each candidate pair in the check list has a foundation and a state. The foundation is the combination of the foundations of the local and remote candidates in the pair. The state is assigned once the check list for each media stream has been computed. There are five potential values that the state can have:

Waiting: A check has not been performed for this pair, and can be performed as soon as it is the highest-priority Waiting pair on the check list.

In-Progress: A check has been sent for this pair, but the transaction is in progress.

Succeeded: A check for this pair was already done and produced a successful result.

Failed: A check for this pair was already done and failed, either never producing any response or producing an unrecoverable failure response.

Frozen: A check for this pair hasn't been performed, and it can't yet be performed until some other check succeeds, allowing this pair to unfreeze and move into the Waiting state.

As ICE runs, the pairs will move between states as shown in Figure 7.

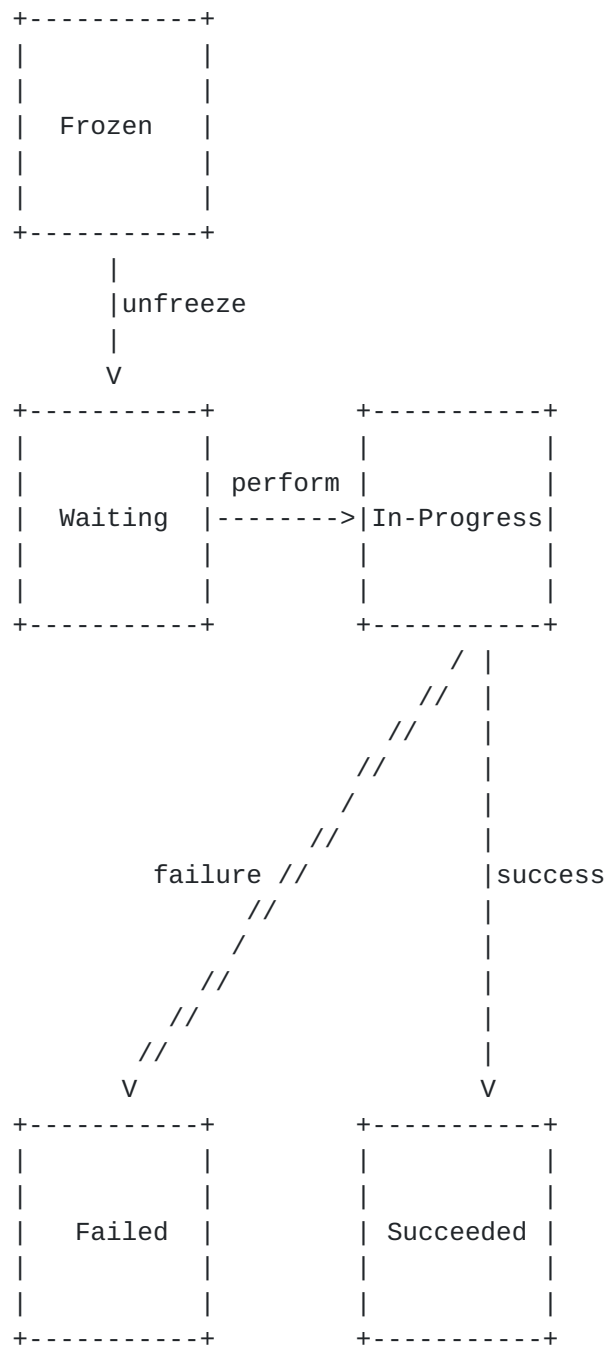


Figure 7: Pair State FSM

The initial states for each pair in a check list are computed by performing the following sequence of steps:

1. The agent sets all of the pairs in each check list to the Frozen state.

2. The agent examines the check list for the first media stream. For that media stream:

   * For all pairs with the same foundation, it sets the state of the pair with the lowest component ID to Waiting. If there is more than one such pair, the one with the highest-priority is used.

One of the check lists will have some number of pairs in the Waiting state, and the other check lists will have all of their pairs in the Frozen state. A check list with at least one pair that is Waiting is called an active check list, and a check list with all pairs Frozen is called a frozen check list.

The check list itself is associated with a state, which captures the state of ICE checks for that media stream. There are three states:

Running: In this state, ICE checks are still in progress for this media stream.

Completed: In this state, ICE checks have produced nominated pairs for each component of the media stream.

Failed: In this state, the ICE checks have not completed successfully for this media stream.

When a check list is first constructed as the consequence of an candidate exchange, it is placed in the Running state.

ICE processing across all media streams also has a state associated with it. This state is equal to Running while ICE processing is under way. The state is Completed when ICE processing is complete and Failed if it failed without success. Rules for transitioning between states are described below.

5.1.4. Scheduling Checks

An agent performs ordinary checks and triggered checks. The generation of both checks is governed by a timer that fires periodically for each media stream. The agent maintains a FIFO queue, called the triggered check queue, which contains candidate pairs for which checks are to be sent at the next available opportunity. When the timer fires, the agent removes the top pair from the triggered check queue, performs a connectivity check on that pair, and sets the state of the candidate pair to In-Progress. If there are no pairs in the triggered check queue, an ordinary check is sent.

Once the agent has computed the check lists as described in Section 5.1.3, it sets a timer for each active check list. The timer fires every Ta*N seconds, where N is the number of active check lists (initially, there is only one active check list). Implementations MAY set the timer to fire less frequently than this. Implementations SHOULD take care to spread out these timers so that they do not fire at the same time for each media stream. Ta and the retransmit timer RTO are computed as described in Section 13. Multiplying by N allows this aggregate check throughput to be split between all active check lists. The first timer fires immediately, so that the agent performs a connectivity check the moment the candidate exchange has been done, followed by the next check Ta seconds later (since there is only one active check list).

When the timer fires and there is no triggered check to be sent, the agent MUST choose an ordinary check as follows:

- Find the highest-priority pair in that check list that is in the Waiting state.
- If there is such a pair:
  - Send a STUN check from the local candidate of that pair to the remote candidate of that pair. The procedures for forming the STUN request for this purpose are described in Section 6.1.2.
  - Set the state of the candidate pair to In-Progress.
- If there is no such pair:
  - Find the highest-priority pair in that check list that is in the Frozen state.
  - If there is such a pair:
    - Unfreeze the pair.
    - Perform a check for that pair, causing its state to transition to In-Progress.
  - If there is no such pair:
    - Terminate the timer for that check list.

To compute the message integrity for the check, the agent uses the remote username fragment and password learned from the candidate information obtained from its peer. The local username fragment is known directly by the agent for its own candidate.

The Initiator performs the ordinary checks on receiving the candidate information from the Peer (responder) and having formed the checklists. On the other hand the responding agent either performs the triggered or ordinary checks as described above.

5.2. Lite Implementation Procedures

Lite implementations skips most of the steps in Section 5 except for verifying the peer's ICE support and determining its role in the ICE processing.

On determining the role for a lite implementation being the controlling agent means selecting a candidate pair based on the ones in the candidate exchange (for IPv4, there is only ever one pair), and then updating the peer with the new candidate information reflecting that selection, when needed (it is never needed for an IPv4-only host). The controlled agent is told which candidate pairs to use for each media stream, and no further candidate updates are needed to signal this information.

6. Performing Connectivity Checks

This section describes how connectivity checks are performed. All ICE implementations are required to be compliant to [RFC5389], as opposed to the older [RFC3489]. However, whereas a full implementation will both generate checks (acting as a STUN client) and receive them (acting as a STUN server), a lite implementation will only receive checks, and thus will only act as a STUN server.

6.1. STUN Client Procedures

These procedures define how an agent sends a connectivity check, whether it is an ordinary or a triggered check. These procedures are only applicable to full implementations.

6.1.1. Creating Permissions for Relayed Candidates

If the connectivity check is being sent using a relayed local candidate, the client MUST create a permission first if it has not already created one previously. It would have created one previously if it had told the TURN server to create a permission for the given relayed candidate towards the IP address of the remote candidate. To create the permission, the agent follows the procedures defined in [RFC5766]. The permission MUST be created towards the IP address of the remote candidate. It is RECOMMENDED that the agent defer creation of a TURN channel until ICE completes, in which case permissions for connectivity checks are normally created using a

CreatePermission request. Once established, the agent MUST keep the permission active until ICE concludes.

6.1.2. Sending the Request

A connectivity check is generated by sending a Binding request from a local candidate to a remote candidate. [RFC5389] describes how Binding requests are constructed and generated. A connectivity check MUST utilize the STUN short-term credential mechanism. Support for backwards compatibility with RFC 3489 MUST NOT be used or assumed with connectivity checks. The FINGERPRINT mechanism MUST be used for connectivity checks.

ICE extends STUN by defining several new attributes, including PRIORITY, USE-CANDIDATE, ICE-CONTROLLED, and ICE-CONTROLLING. These new attributes are formally defined in Section 16.1, and their usage is described in the subsections below. These STUN extensions are applicable only to connectivity checks used for ICE.

6.1.2.1. PRIORITY

An agent MUST include the PRIORITY attribute in its Binding request. The attribute MUST be set equal to the priority that would be assigned, based on the algorithm in Section 4.1.2, to a peer reflexive candidate, should one be learned as a consequence of this check (see Section 6.1.3.2.1 for how peer reflexive candidates are learned). This priority value will be computed identically to how the priority for the local candidate of the pair was computed, except that the type preference is set to the value for peer reflexive candidate types.

6.1.2.2. USE-CANDIDATE

The controlling agent includes the USE-CANDIDATE attribute in order to nominate a candidate pair Section 7.1.1. The controlled agent MUST NOT include the USE-CANDIDATE attribute in its Binding request.

6.1.2.3. ICE-CONTROLLED and ICE-CONTROLLING

The agent MUST include the ICE-CONTROLLED attribute in the request if it is in the controlled role, and MUST include the ICE-CONTROLLING attribute in the request if it is in the controlling role. The content of either attribute MUST be the tie-breaker that was determined in Section 5.1.2. These attributes are defined fully in Section 16.1.

6.1.2.4. Forming Credentials

A Binding request serving as a connectivity check MUST utilize the STUN short-term credential mechanism. The username for the credential is formed by concatenating the username fragment provided by the peer with the username fragment of the agent sending the request, separated by a colon (":"). The password is equal to the password provided by the peer. For example, consider the case where agent L is the initiating , agent and agent R is the responding agent. Agent L included a username fragment of LFRAG for its candidates and a password of LPASS. Agent R provided a username fragment of RFRAG and a password of RPASS. A connectivity check from L to R utilizes the username RFRAG:LFRAG and a password of RPASS. A connectivity check from R to L utilizes the username LFRAG:RFRAG and a password of LPASS. The responses utilize the same usernames and passwords as the requests (note that the USERNAME attribute is not present in the response).

6.1.2.5. DiffServ Treatment

If the agent is using Diffserv Codepoint markings [RFC2475] in its media packets, it SHOULD apply those same markings to its connectivity checks.

6.1.3. Processing the Response

When a Binding response is received, it is correlated to its Binding request using the transaction ID, as defined in [RFC5389], which then ties it to the candidate pair for which the Binding request was sent. This section defines additional procedures for processing Binding responses specific to this usage of STUN.

6.1.3.1. Failure Cases

If the STUN transaction generates a 487 (Role Conflict) error response, the agent checks whether it included the ICE-CONTROLLED or ICE-CONTROLLING attribute in the Binding request. If the request contained the ICE-CONTROLLED attribute, the agent MUST switch to the controlling role if it has not already done so. If the request contained the ICE-CONTROLLING attribute, the agent MUST switch to the controlled role if it has not already done so. Once it has switched, the agent MUST enqueue the candidate pair whose check generated the 487 into the triggered check queue. The state of that pair is set to Waiting. When the triggered check is sent, it will contain an ICE-CONTROLLING or ICE-CONTROLLED attribute reflecting its new role. Note, however, that the tie-breaker value MUST NOT be reselected.

A change in roles will require an agent to recompute pair priorities (Section 5.1.3.2), since those priorities are a function of controlling and controlled roles. The change in role will also impact whether the agent is responsible for selecting nominated pairs and generating updated candidate information for sharing upon conclusion of ICE.

Agents MAY support receipt of ICMP errors for connectivity checks. If the STUN transaction generates an ICMP error, the agent sets the state of the pair to Failed. If the STUN transaction generates a STUN error response that is unrecoverable (as defined in [RFC5389]) or times out, the agent sets the state of the pair to Failed.

The agent MUST check that the source IP address and port of the response equal the destination IP address and port to which the Binding request was sent, and that the destination IP address and port of the response match the source IP address and port from which the Binding request was sent. In other words, the source and destination transport addresses in the request and responses are symmetric. If they are not symmetric, the agent sets the state of the pair to Failed.

6.1.3.2. Success Cases

A check is considered to be a success if all of the following are true:

- The STUN transaction generated a success response.
- The source IP address and port of the response equals the destination IP address and port to which the Binding request was sent.
- The destination IP address and port of the response match the source IP address and port from which the Binding request was sent.

6.1.3.2.1. Discovering Peer Reflexive Candidates

The agent checks the mapped address from the STUN response. If the transport address does not match any of the local candidates that the agent knows about, the mapped address represents a new candidate -- a peer reflexive candidate. Like other candidates, it has a type, base, priority, and foundation. They are computed as follows:

- Its type is equal to peer reflexive.

- Its base is set equal to the local candidate of the candidate pair from which the STUN check was sent.

- Its priority is set equal to the value of the PRIORITY attribute in the Binding request.

- Its foundation is selected as described in Section 4.1.1.3.

This peer reflexive candidate is then added to the list of local candidates for the media stream. Its username fragment and password are the same as all other local candidates for that media stream. However, the peer reflexive candidate is not paired with other remote candidates. This is not necessary; a valid pair will be generated from it momentarily based on the procedures in Section 6.1.3.2.2. If an agent wishes to pair the peer reflexive candidate with other remote candidates besides the one in the valid pair that will be generated, the agent MAY generate an update the peer with the candidate information that includes the peer reflexive candidate. This will cause it to be paired with all other remote candidates.

6.1.3.2.2. Constructing a Valid Pair

The agent constructs a candidate pair whose local candidate equals the mapped address of the response, and whose remote candidate equals the destination address to which the request was sent. This is called a valid pair, since it has been validated by a STUN connectivity check. The valid pair may equal the pair that generated the check, may equal a different pair in the check list, or may be a pair not currently on any check list. If the pair equals the pair that generated the check or is on a check list currently, it is also added to the VALID LIST, which is maintained by the agent for each media stream. This list is empty at the start of ICE processing, and fills as checks are performed, resulting in valid candidate pairs.

It will be very common that the pair will not be on any check list. Recall that the check list has pairs whose local candidates are never server reflexive; those pairs had their local candidates converted to the base of the server reflexive candidates, and then pruned if they were redundant. When the response to the STUN check arrives, the mapped address will be reflexive if there is a NAT between the two. In that case, the valid pair will have a local candidate that doesn't match any of the pairs in the check list.

If the pair is not on any check list, the agent computes the priority for the pair based on the priority of each candidate, using the algorithm in Section 5.1.3. The priority of the local candidate depends on its type. If it is not peer reflexive, it is equal to the priority signaled for that candidate in the candidate exchange. If

it is peer reflexive, it is equal to the PRIORITY attribute the agent placed in the Binding request that just completed. The priority of the remote candidate is taken from the candidate information of the peer. If the candidate does not appear there, then the check must have been a triggered check to a new remote candidate. In that case, the priority is taken as the value of the PRIORITY attribute in the Binding request that triggered the check that just completed. The pair is then added to the VALID LIST.

6.1.3.2.3. Updating Pair States

The agent sets the state of the pair that *generated* the check to Succeeded. Note that, the pair which *generated* the check may be different than the valid pair constructed in Section 6.1.3.2.2 as a consequence of the response. The success of this check might also cause the state of other checks to change as well. The agent MUST perform the following two steps:

1. The agent changes the states for all other Frozen pairs for the same media stream and same foundation to Waiting. Typically, but not always, these other pairs will have different component IDs.

2. If there is a pair in the valid list for every component of this media stream (where this is the actual number of components being used, in cases where the number of components signaled in the candidate exchange differs from initiating to responding agent), the success of this check may unfreeze checks for other media streams. Note that this step is followed not just the first time the valid list under consideration has a pair for every component, but every subsequent time a check succeeds and adds yet another pair to that valid list. The agent examines the check list for each other media stream in turn:

   * If the check list is active, the agent changes the state of all Frozen pairs in that check list whose foundation matches a pair in the valid list under consideration to Waiting.

   * If the check list is frozen, and there is at least one pair in the check list whose foundation matches a pair in the valid list under consideration, the state of all pairs in the check list whose foundation matches a pair in the valid list under consideration is set to Waiting. This will cause the check list to become active, and ordinary checks will begin for it, as described in Section 5.1.4.

   * If the check list is frozen, and there are no pairs in the check list whose foundation matches a pair in the valid list under consideration, the agent

+ groups together all of the pairs with the same foundation, and

+ for each group, sets the state of the pair with the lowest component ID to Waiting. If there is more than one such pair, the one with the highest-priority is used.

6.1.3.2.4. Updating the Nominated Flag

If the agent was a controlling agent, and it had included a USE-CANDIDATE attribute in the Binding request, the valid pair generated from that check has its nominated flag set to true. This flag indicates that this valid pair SHOULD be used for media, unless the sending agent detects that the candidiate pair does not work. This concludes the ICE processing for this media stream or all media streams; see Section 7.

If the agent is the controlled agent, the response may be the result of a triggered check that was sent in response to a request that itself had the USE-CANDIDATE attribute. This case is described in Section 6.2.1.5, and may now result in setting the nominated flag for the pair learned from the original request.

An agent MUST NOT select a candidate pair until it has sent a Binding request, and received the corresponding Binding response, associated with the candidiate pair.

6.1.3.3. Check List and Timer State Updates

Regardless of whether the check was successful or failed, the completion of the transaction may require updating of check list and timer states.

If all of the pairs in the check list are now either in the Failed or Succeeded state:

o If there is not a pair in the valid list for each component of the media stream, the state of the check list is set to Failed.

o For each frozen check list, the agent

  * groups together all of the pairs with the same foundation, and

  * for each group, sets the state of the pair with the lowest component ID to Waiting. If there is more than one such pair, the one with the highest-priority is used.

If none of the pairs in the check list are in the Waiting or Frozen state, the check list is no longer considered active, and will not count towards the value of N in the computation of timers for ordinary checks as described in Section 5.1.4.

6.2. STUN Server Procedures

An agent MUST be prepared to receive a Binding request on the base of each candidate it included in its most recent candidate exchange. This requirement holds even if the peer is a lite implementation.

The agent MUST use the short-term credential mechanism (i.e., the MESSAGE-INTEGRITY attribute) to authenticate the request and perform a message integrity check. Likewise, the short-term credential mechanism MUST be used for the response. The agent MUST consider the username to be valid if it consists of two values separated by a colon, where the first value is equal to the username fragment generated by the agent in an candidate exchange for a session in-progress. It is possible (and in fact very likely) that the initiating agent will receive a Binding request prior to receiving the candidates from its peer. If this happens, the agent MUST immediately generate a response (including computation of the mapped address as described in Section 6.2.1.2). The agent has sufficient information at this point to generate the response; the password from the peer is not required. Once the answer is received, it MUST proceed with the remaining steps required, namely, Section 6.2.1.3, Section 6.2.1.4, and Section 6.2.1.5 for full implementations. In cases where multiple STUN requests are received before the answer, this may cause several pairs to be queued up in the triggered check queue.

An agent MUST NOT utilize the ALTERNATE-SERVER mechanism, and MUST NOT support the backwards-compatibility mechanisms to RFC 3489. It MUST utilize the FINGERPRINT mechanism.

If the agent is using Diffserv Codepoint markings [RFC2475] in its media packets, it SHOULD apply those same markings to its responses to Binding requests. The same would apply to any layer 2 markings the endpoint might be applying to media packets.

6.2.1. Additional Procedures for Full Implementations

This subsection defines the additional server procedures applicable to full implementations.

6.2.1.1. Detecting and Repairing Role Conflicts

Normally, the rules for selection of a role in Section 5.1.2 will result in each agent selecting a different role -- one controlling and one controlled. However, in unusual call flows, typically utilizing third party call control, it is possible for both agents to select the same role. This section describes procedures for checking for this case and repairing it. These procedures apply only to usages of ICE that require conflict resolution. The usage document MUST specify whether this mechanism is needed.

An agent MUST examine the Binding request for either the ICE-CONTROLLING or ICE-CONTROLLED attribute. It MUST follow these procedures:

- If neither ICE-CONTROLLING nor ICE-CONTROLLED is present in the request, the peer agent may have implemented a previous version of this specification. There may be a conflict, but it cannot be detected.

- If the agent is in the controlling role, and the ICE-CONTROLLING attribute is present in the request:

  * If the agent's tie-breaker is larger than or equal to the contents of the ICE-CONTROLLING attribute, the agent generates a Binding error response and includes an ERROR-CODE attribute with a value of 487 (Role Conflict) but retains its role.

  * If the agent's tie-breaker is less than the contents of the ICE-CONTROLLING attribute, the agent switches to the controlled role.

- If the agent is in the controlled role, and the ICE-CONTROLLED attribute is present in the request:

  * If the agent's tie-breaker is larger than or equal to the contents of the ICE-CONTROLLED attribute, the agent switches to the controlling role.

  * If the agent's tie-breaker is less than the contents of the ICE-CONTROLLED attribute, the agent generates a Binding error response and includes an ERROR-CODE attribute with a value of 487 (Role Conflict) but retains its role.

- If the agent is in the controlled role and the ICE-CONTROLLING attribute was present in the request, or the agent was in the controlling role and the ICE-CONTROLLED attribute was present in the request, there is no conflict.

A change in roles will require an agent to recompute pair priorities (Section 5.1.3.2), since those priorities are a function of controlling and controlled roles. The change in role will also impact whether the agent is responsible for selecting nominated pairs and initiating exchange with updated candidate information upon conclusion of ICE.

The remaining sections in Section 6.2.1 are followed if the server generated a successful response to the Binding request, even if the agent changed roles.

6.2.1.2. Computing Mapped Address

For requests being received on a relayed candidate, the source transport address used for STUN processing (namely, generation of the XOR-MAPPED-ADDRESS attribute) is the transport address as seen by the TURN server. That source transport address will be present in the XOR-PEER-ADDRESS attribute of a Data Indication message, if the Binding request was delivered through a Data Indication. If the Binding request was delivered through a ChannelData message, the source transport address is the one that was bound to the channel.

6.2.1.3. Learning Peer Reflexive Candidates

If the source transport address of the request does not match any existing remote candidates, it represents a new peer reflexive remote candidate. This candidate is constructed as follows:

- The priority of the candidate is set to the PRIORITY attribute from the request.
- The type of the candidate is set to peer reflexive.
- The foundation of the candidate is set to an arbitrary value, different from the foundation for all other remote candidates. If any subsequent candidate exchanges contain this peer reflexive candidate, it will signal the actual foundation for the candidate.
- The component ID of this candidate is set to the component ID for the local candidate to which the request was sent.

This candidate is added to the list of remote candidates. However, the agent does not pair this candidate with any local candidates.

6.2.1.4. Triggered Checks

Next, the agent constructs a pair whose local candidate is equal to the transport address on which the STUN request was received, and a remote candidate equal to the source transport address where the request came from (which may be the peer reflexive remote candidate that was just learned). The local candidate will either be a host candidate (for cases where the request was not received through a relay) or a relayed candidate (for cases where it is received through a relay). The local candidate can never be a server reflexive candidate. Since both candidates are known to the agent, it can obtain their priorities and compute the candidate pair priority. This pair is then looked up in the check list. There can be one of several outcomes:

- If the pair is already on the check list:
  - If the state of that pair is Waiting or Frozen, a check for that pair is enqueued into the triggered check queue if not already present.
  - If the state of that pair is In-Progress, the agent cancels the in-progress transaction. Cancellation means that the agent will not retransmit the request, will not treat the lack of response to be a failure, but will wait the duration of the transaction timeout for a response. In addition, the agent MUST create a new connectivity check for that pair (representing a new STUN Binding request transaction) by enqueueing the pair in the triggered check queue. The state of the pair is then changed to Waiting.
  - If the state of the pair is Failed, it is changed to Waiting and the agent MUST create a new connectivity check for that pair (representing a new STUN Binding request transaction), by enqueueing the pair in the triggered check queue.
  - If the state of that pair is Succeeded, nothing further is done.

These steps are done to facilitate rapid completion of ICE when both agents are behind NAT.

- If the pair is not already on the check list:
  - The pair is inserted into the check list based on its priority.
  - Its state is set to Waiting.

* The pair is enqueued into the triggered check queue.

When a triggered check is to be sent, it is constructed and processed as described in Section 6.1.2. These procedures require the agent to know the transport address, username fragment, and password for the peer. The username fragment for the remote candidate is equal to the part after the colon of the USERNAME in the Binding request that was just received. Using that username fragment, the agent can check the candidates received from its peer (there may be more than one in cases of forking), and find this username fragment. The corresponding password is then selected.

6.2.1.5. Updating the Nominated Flag

If the Binding request received by the agent had the USE-CANDIDATE attribute set, and the agent is in the controlled role, the agent looks at the state of the pair computed in Section 6.2.1.4:

- If the state of this pair is Succeeded, it means that the check generated by this pair produced a successful response. This would have caused the agent to construct a valid pair when that success response was received (see Section 6.1.3.2.2). The agent now sets the nominated flag in the valid pair to true. This may end ICE processing for this media stream; see Section 7.

- If the state of this pair is In-Progress, if its check produces a successful result, the resulting valid pair has its nominated flag set when the response arrives. This may end ICE processing for this media stream when it arrives; see Section 7.

6.2.2. Additional Procedures for Lite Implementations

If the check that was just received contained a USE-CANDIDATE attribute, the agent constructs a candidate pair whose local candidate is equal to the transport address on which the request was received, and whose remote candidate is equal to the source transport address of the request that was received. This candidate pair is assigned an arbitrary priority, and placed into a list of valid candidates called the valid list. The agent sets the nominated flag for that pair to true. ICE processing is considered complete for a media stream if the valid list contains a candidate pair for each component.

7. Concluding ICE Processing

This section describes how an agent completes ICE.

7.1. Procedures for Full Implementations

Concluding ICE involves nominating pairs by the controlling agent and updating of state machinery.

7.1.1. Nominating Pairs

When nominating, the controlling agent lets some number of checks complete, each of which omit the USE-CANDIDATE attribute. Once one or more checks complete successfully for a component of a media stream, valid pairs are generated and added to the valid list. The agent lets the checks continue until some stopping criterion is met, and then picks amongst the valid pairs based on an evaluation criterion. The criteria for stopping the checks and for evaluating the valid pairs is entirely a matter of local optimization.

When the controlling agent selects the valid pair, it repeats the check that produced this valid pair (by enqueueing the pair that generated the check into the triggered check queue), this time with the USE-CANDIDATE attribute. This check should succeed (since the previous did), causing the nominated flag of that and only that pair to be set. Consequently, there will be only a single nominated pair in the valid list for each component, and when the state of the check list moves to completed, that exact pair is selected by ICE for sending and receiving media for that component.

The controlling agent has control over the stopping and selection criteria for checks. The only requirement is that the agent MUST eventually pick one and only one candidate pair and generate a check for that pair with the USE-CANDIDATE attribute present.

The controlled agent SHOULD select the nominated candidate pair if the agent is receiving Binding responses associated with that candidiate pair. Before the agent has received Binding responses associated with the candidiate pair, the agent can send media on any candidiate for which it has received Binding responses. If more than one candidate pair is nominated by the controlling agent, the controlled agent SHOULD select the candidate pair with the highest priority.

NOTE: A controlling agent that does not support this speification (i.e. it is implemented according to RFC 5245) might nominate more than one candidiate pair. This was referred to as aggressive nomination in RFC 5245. The usage of the 'ice2' ice option by endpoints supporting this specifcation should prevent such controlling agents from using aggressive nomination.

7.1.2. Updating States

For both controlling and controlled agents, the state of ICE processing depends on the presence of nominated candidate pairs in the valid list and on the state of the check list. Note that, at any time, more than one of the following cases can apply:

- If there are no nominated pairs in the valid list for a media stream and the state of the check list is Running, ICE processing continues.

- If there is at least one nominated pair in the valid list for a media stream and the state of the check list is Running:

  * The agent MUST remove all Waiting and Frozen pairs in the check list and triggered check queue for the same component as the nominated pairs for that media stream.

  * If an In-Progress pair in the check list is for the same component as a nominated pair, the agent SHOULD cease retransmissions for its check if its pair priority is lower than the lowest-priority nominated pair for that component.

- Once there is at least one nominated pair in the valid list for every component of at least one media stream and the state of the check list is Running:

  * The agent MUST change the state of processing for its check list for that media stream to Completed.

  * The agent MUST continue to respond to any checks it may still receive for that media stream, and MUST perform triggered checks if required by the processing of Section 6.2.

  * The agent MUST continue retransmitting any In-Progress checks for that check list.

  * The agent MAY begin transmitting media for this media stream as described in Section 11.1.

- Once the state of each check list is Completed:

  * The agent sets the state of ICE processing overall to Completed.

- If the state of the check list is Failed, ICE has not been able to complete for this media stream. The correct behavior depends on the state of the check lists for other media streams:

* If all check lists are Failed, ICE processing overall is considered to be in the Failed state, and the agent SHOULD consider the session a failure, SHOULD NOT restart ICE, and the controlling agent SHOULD terminate the entire session.

* If at least one of the check lists for other media streams is Completed, the controlling agent SHOULD remove the failed media stream from the session while sending updated candidate list to its peer.

* If none of the check lists for other media streams are Completed, but at least one is Running, the agent SHOULD let ICE continue.

7.2. Procedures for Lite Implementations

Concluding ICE for a lite implementation is relatively straightforward. There are two cases to consider:

The implementation is lite, and its peer is full.

The implementation is lite, and its peer is lite.

The effect of ICE concluding is that the agent can free any allocated host candidates that were not utilized by ICE, as described in Section 7.3.

7.2.1. Peer Is Full

In this case, the agent will receive connectivity checks from its peer. When an agent has received a connectivity check that includes the USE-CANDIDATE attribute for each component of a media stream, the state of ICE processing for that media stream moves from Running to Completed. When the state of ICE processing for all media streams is Completed, the state of ICE processing overall is Completed.

The lite implementation will never itself determine that ICE processing has failed for a media stream; rather, the full peer will make that determination and then remove or restart the failed media stream as part of subsequent candidate exchange process.

7.2.2. Peer Is Lite

Once the candidate exchange has completed, both agents examine their candidates and those of its peer. For each media stream, each agent pairs up its own candidates with the candidates of its peer for that media stream. Two candidates are paired up when they are for the same component, utilize the same transport protocol (UDP in this

specification), and are from the same IP address family (IPv4 or IPv6).

- If there is a single pair per component, that pair is added to the Valid list. If all of the components for a media stream had one pair, the state of ICE processing for that media stream is set to Completed. If all media streams are Completed, the state of ICE processing is set to Completed overall. This will always be the case for implementations that are IPv4-only.

- If there is more than one pair per component:

  * The agent MUST select a pair based on local policy. Since this case only arises for IPv6, it is RECOMMENDED that an agent follow the procedures of RFC 6724 [RFC6724] to select a single pair.

  * The agent adds the selected pair for each component to the valid list. As described in Section 11.1, this will permit media to begin flowing. However, it is possible (and in fact likely) that both agents have chosen different pairs.

  * To reconcile this, the controlling agent MUST send updated candidate list which will include the remote-candidates attribute.

  * The agent MUST NOT update the state of ICE processing until after the candidate exchange completes. Then the controlling agent MUST change the state of ICE processing to Completed for all media streams, and the state of ICE processing overall to Completed.

7.3. Freeing Candidates

7.3.1. Full Implementation Procedures

The procedures in Section 7 require that an agent continue to listen for STUN requests and continue to generate triggered checks for a media stream, even once processing for that stream completes. The rules in this section describe when it is safe for an agent to cease sending or receiving checks on a candidate that was not selected by ICE, and then free the candidate.

7.3.2. Lite Implementation Procedures

A lite implementation MAY free candidates not selected by ICE as soon as ICE processing has reached the Completed state for all peers for all media streams using those candidates.

8. ICE Restarts

An agent MAY restart ICE processing for an existing media stream. An ICE restart, as the name implies, will cause all previous states of ICE processing to be flushed and checks to start anew. The only difference between an ICE restart and a brand new media session is that, during the restart, media can continue to be sent to the previously validated pair.

An agent MUST restart ICE for a media stream if:

- The candidate(s) is being generated for the purposes of changing the target of the media stream. In other words, if an agent wants to generate an updated candidate information that, had ICE not been in use, would result in a new value for the destination of a media component.

- An agent is changing its implementation level. This typically only happens in third party call control use cases, where the entity performing the signaling is not the entity receiving the media, and it has changed the target of media mid-session to another entity that has a different ICE implementation.

To restart ICE, an agent MUST change both the password and the user name fragment for the media stream when exchanging the candidates. The new candidate set MAY include some, none, or all of the previous candidates for that stream and MAY include a totally new set of candidates.

9. ICE Option

This section defines a new ICE option, 'ice2'. The ICE option indicates that the ICE agent that includes it (in an ice-options attribute) is compliant to this specification. For example, the ICE agent will not use the aggressive nomination procedure defined in [RFC5245].

An ICE agent compliant to this specification MUST inform the peer about the compliance using the 'ice2' ICE option.

NOTE: The encoding of the 'ice2' ICE option, and the message(s) used to carry it to the peer, are protocol specific. The encoding for the Session Description Protocol (SDP) [RFC4566] is defined in [I-D.ietf-mmusic-ice-sip-sdp].

10. Keepalives

All endpoints MUST send keepalives for each media session. These keepalives serve the purpose of keeping NAT bindings alive for the media session. These keepalives MUST be sent even if ICE is not being utilized for the session at all. The keepalive SHOULD be sent using a format that is supported by its peer. ICE endpoints allow for STUN-based keepalives for UDP streams, and as such, STUN keepalives MUST be used when an agent is a full ICE implementation and is communicating with a peer that supports ICE (lite or full). If the peer does not support ICE, the choice of a packet format for keepalives is a matter of local implementation. A format that allows packets to easily be sent in the absence of actual media content is RECOMMENDED. Examples of formats that readily meet this goal are RTP No-Op [I-D.ietf-avt-rtp-no-op], and in cases where both sides support it, RTP comfort noise [RFC3389]. If the peer doesn't support any formats that are particularly well suited for keepalives, an agent SHOULD send RTP packets with an incorrect version number, or some other form of error that would cause them to be discarded by the peer.

If there has been no packet sent on the candidate pair ICE is using for a media component for Tr seconds (where packets include those defined for the component (RTP or RTCP) and previous keepalives), an agent MUST generate a keepalive on that pair. Tr SHOULD be configurable and SHOULD have a default of 15 seconds. Tr MUST NOT be configured to less than 15 seconds. Alternatively, if an agent has a dynamic way to discover the binding lifetimes of the intervening NATs, it can use that value to determine Tr. Administrators deploying ICE in more controlled networking environments SHOULD set Tr to the longest duration possible in their environment.

If STUN is being used for keepalives, a STUN Binding Indication is used [RFC5389]. The Indication MUST NOT utilize any authentication mechanism. It SHOULD contain the FINGERPRINT attribute to aid in demultiplexing, but SHOULD NOT contain any other attributes. It is used solely to keep the NAT bindings alive. The Binding Indication is sent using the same local and remote candidates that are being used for media. Though Binding Indications are used for keepalives, an agent MUST be prepared to receive a connectivity check as well. If a connectivity check is received, a response is generated as discussed in [RFC5389], but there is no impact on ICE processing otherwise.

An agent MUST begin the keepalive processing once ICE has selected candidates for usage with media, or media begins to flow, whichever happens first. Keepalives end once the session terminates or the media stream is removed.

11. Media Handling

11.1. Sending Media

Procedures for sending media differ for full and lite implementations.

11.1.1. Procedures for Full Implementations

Agents always send media using a candidate pair, called the selected candidate pair. An agent will send media to the remote candidate in the selected pair (setting the destination address and port of the packet equal to that remote candidate), and will send it from the local candidate of the selected pair. When the local candidate is server or peer reflexive, media is originated from the base. Media sent from a relayed candidate is sent from the base through that TURN server, using procedures defined in [RFC5766].

If the local candidate is a relayed candidate, it is RECOMMENDED that an agent create a channel on the TURN server towards the remote candidate. This is done using the procedures for channel creation as defined in Section 11 of [RFC5766].

The selected pair for a component of a media stream is:

- empty if the state of the check list for that media stream is Running, and there is no previous selected pair for that component due to an ICE restart

- equal to the previous selected pair for a component of a media stream if the state of the check list for that media stream is Running, and there was a previous selected pair for that component due to an ICE restart

- equal to the highest-priority nominated pair for that component in the valid list if the state of the check list is Completed

If the selected pair for at least one component of a media stream is empty, an agent MUST NOT send media for any component of that media stream. If the selected pair for each component of a media stream has a value, an agent MAY send media for all components of that media stream.

11.1.2. Procedures for Lite Implementations

A lite implementation MUST NOT send media until it has a Valid list that contains a candidate pair for each component of that media stream. Once that happens, the agent MAY begin sending media

packets. To do that, it sends media to the remote candidate in the pair (setting the destination address and port of the packet equal to that remote candidate), and will send it from the local candidate.

11.1.3. Procedures for All Implementations

ICE has interactions with jitter buffer adaptation mechanisms. An RTP stream can begin using one candidate, and switch to another one, though this happens rarely with ICE. The newer candidate may result in RTP packets taking a different path through the network -- one with different delay characteristics. As discussed below, agents are encouraged to re-adjust jitter buffers when there are changes in source or destination address of media packets. Furthermore, many audio codecs use the marker bit to signal the beginning of a talkspurt, for the purposes of jitter buffer adaptation. For such codecs, it is RECOMMENDED that the sender set the marker bit [RFC3550] when an agent switches transmission of media from one candidate pair to another.

11.2. Receiving Media

ICE implementations MUST be prepared to receive media on each component on any candidates provided for that component in the most recent candidate exchange (in the case of RTP, this would include both RTP and RTCP if candidates were provided for both).

It is RECOMMENDED that, when an agent receives an RTP packet with a new source or destination IP address for a particular media stream, that the agent re-adjust its jitter buffers.

RFC 3550 [RFC3550] describes an algorithm in Section 8.2 for detecting synchronization source (SSRC) collisions and loops. These algorithms are based, in part, on seeing different source transport addresses with the same SSRC. However, when ICE is used, such changes will sometimes occur as the media streams switch between candidates. An agent will be able to determine that a media stream is from the same peer as a consequence of the STUN exchange that proceeds media transmission. Thus, if there is a change in source transport address, but the media packets come from the same peer agent, this SHOULD NOT be treated as an SSRC collision.

12. Extensibility Considerations

This specification makes very specific choices about how both agents in a session coordinate to arrive at the set of candidate pairs that are selected for media. It is anticipated that future specifications will want to alter these algorithms, whether they are simple changes like timer tweaks or larger changes like a revamp of the priority

algorithm. When such a change is made, providing interoperability between the two agents in a session is critical.

First, ICE provides the ice-options attribute. Each extension or change to ICE is associated with a token. When an agent supporting such an extension or change triggers candidate exchange, it MUST include the token for that extension in this attribute. This allows each side to know what the other side is doing. This attribute MUST NOT be present if the agent doesn't support any ICE extensions or changes.

One of the complications in achieving interoperability is that ICE relies on a distributed algorithm running on both agents to converge on an agreed set of candidate pairs. If the two agents run different algorithms, it can be difficult to guarantee convergence on the same candidate pairs. The regular nomination procedure described in Section 7 eliminates some of the tight coordination by delegating the selection algorithm completely to the controlling agent. Consequently, when a controlling agent is communicating with a peer that supports options it doesn't know about, the agent MUST run a regular nomination algorithm. When regular nomination is used, ICE will converge perfectly even when both agents use different pair prioritization algorithms. One of the keys to such convergence is triggered checks, which ensure that the nominated pair is validated by both agents. Consequently, any future ICE enhancements MUST preserve triggered checks.

ICE is also extensible to other media streams beyond RTP, and for transport protocols beyond UDP. Extensions to ICE for non-RTP media streams need to specify how many components they utilize, and assign component IDs to them, starting at 1 for the most important component ID. Specifications for new transport protocols must define how, if at all, various steps in the ICE processing differ from UDP.

13. Setting Ta and RTO

During the gathering phase of ICE (Section 4.1.1) and while ICE is performing connectivity checks (Section 6), an agent sends STUN and TURN transactions. These transactions are paced at a rate of one every Ta milliseconds, and utilize a specific RTO. This section describes how the values of Ta and RTO are computed. This computation depends on whether ICE is being used with a real-time media stream (such as RTP) or something else. When ICE is used for a stream with a known maximum bandwidth, the computation in Section 13.1 MAY be followed to rate-control the ICE exchanges. For all other streams, the computation in Section 13.2 MUST be followed.

13.1. Real-time Media Streams

The values of RTO and Ta change during the lifetime of ICE processing. One set of values applies during the gathering phase, and the other, for connectivity checks.

The value of Ta SHOULD be configurable, and SHOULD have a default of:

For each media stream i:
Ta_i = (stun_packet_size / rtp_packet_size) * rtp_ptime

$$Ta = MAX\left(20ms, \frac{1}{\sum_{i=1}^{k} \frac{1}{Ta_i}}\right)$$

where k is the number of media streams. During the gathering phase, Ta is computed based on the number of media streams the agent has indicated in the candidate information, and the RTP packet size and RTP ptime are those of the most preferred codec for each media stream. Once the candidate exchange is completed, the agent recomputes Ta to pace the connectivity checks. In that case, the value of Ta is based on the number of media streams that will actually be used in the session, and the RTP packet size and RTP ptime are those of the most preferred codec with which the agent will send.

In addition, the retransmission timer for the STUN transactions, RTO, defined in [RFC5389], SHOULD be configurable and during the gathering phase, SHOULD have a default of:

$$RTO = MAX(100ms, Ta * (\text{number of pairs}))$$

where the number of pairs refers to the number of pairs of candidates with STUN or TURN servers.

For connectivity checks, RTO SHOULD be configurable and SHOULD have a default of:

RTO = MAX (100ms, Ta*N * (Num-Waiting + Num-In-Progress))

where Num-Waiting is the number of checks in the check list in the Waiting state, and Num-In-Progress is the number of checks in the In-Progress state. Note that the RTO will be different for each transaction as the number of checks in the Waiting and In-Progress states change.

These formulas are aimed at causing STUN transactions to be paced at the same rate as media. This ensures that ICE will work properly under the same network conditions needed to support the media as well. See Appendix B.1 for additional discussion and motivations. Because of this pacing, it will take a certain amount of time to obtain all of the server reflexive and relayed candidates. Implementations should be aware of the time required to do this, and if the application requires a time budget, limit the number of candidates that are gathered.

The formulas result in a behavior whereby an agent will send its first packet for every single connectivity check before performing a retransmit. This can be seen in the formulas for the RTO (which represents the retransmit interval). Those formulas scale with N, the number of checks to be performed. As a result of this, ICE maintains a nicely constant rate, but becomes more sensitive to packet loss. The loss of the first single packet for any connectivity check is likely to cause that pair to take a long time to be validated, and instead, a lower-priority check (but one for which there was no packet loss) is much more likely to complete first. This results in ICE performing sub-optimally, choosing lower-priority pairs over higher-priority pairs. Implementors should be aware of this consequence, but still should utilize the timer values described here.

13.2. Non-real-time Sessions

In cases where ICE is used to establish some kind of session that is not real time, and has no fixed rate associated with it that is known to work on the network in which ICE is deployed, Ta and RTO revert to more conservative values. Ta SHOULD be configurable, SHOULD have a default of 500 ms, and MUST NOT be configurable to be less than 500 ms.

If other Ta value than the default is used, the agent MUST indicate the value it prefers to use in the ICE exchange. Both agents MUST use the higher out of the two proposed values.

In addition, the retransmission timer for the STUN transactions, RTO, SHOULD be configurable and during the gathering phase, SHOULD have a default of:

RTO = MAX (500ms, Ta * (number of pairs))

where the number of pairs refers to the number of pairs of candidates with STUN or TURN servers.

For connectivity checks, RTO SHOULD be configurable and SHOULD have a default of:

RTO = MAX (500ms, Ta*N * (Num-Waiting + Num-In-Progress))

14. Example

The example is based on the simplified topology of Figure 8.

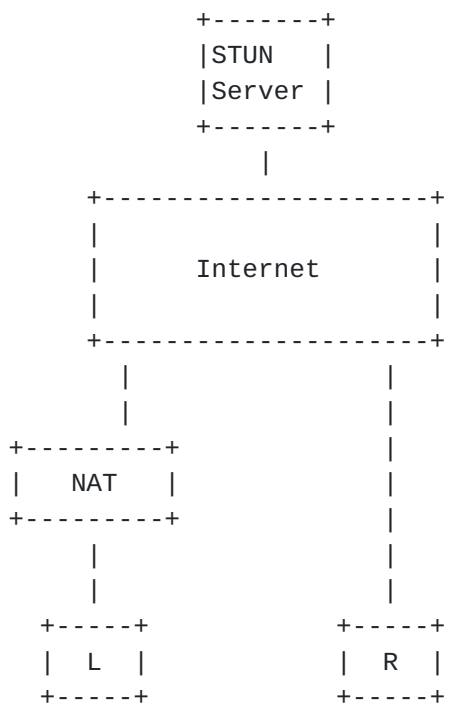


Figure 8: Example Topology

Two agents, L and R, are using ICE. Both are full-mode ICE implementations and use aggressive nomination when they are controlling. Both agents have a single IPv4 address. For agent L, it is 10.0.1.1 in private address space [RFC1918], and for agent R,

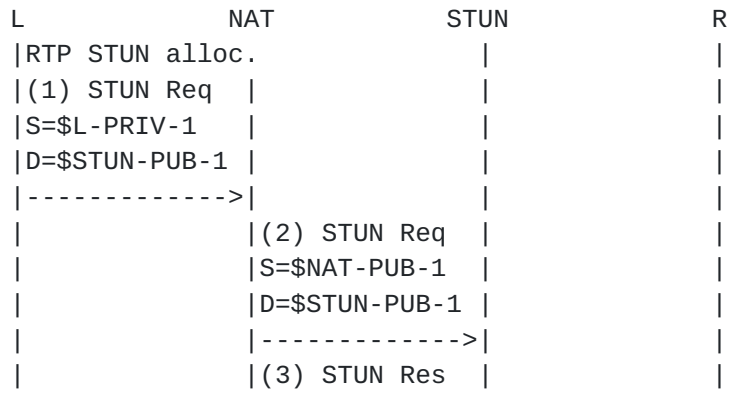
192.0.2.1 on the public Internet. Both are configured with the same STUN server (shown in this example for simplicity, although in practice the agents do not need to use the same STUN server), which is listening for STUN Binding requests at an IP address of 192.0.2.2 and port 3478. TURN servers are not used in this example. Agent L is behind a NAT, and agent R is on the public Internet. The NAT has an endpoint independent mapping property and an address dependent filtering property. The public side of the NAT has an IP address of 192.0.2.3.

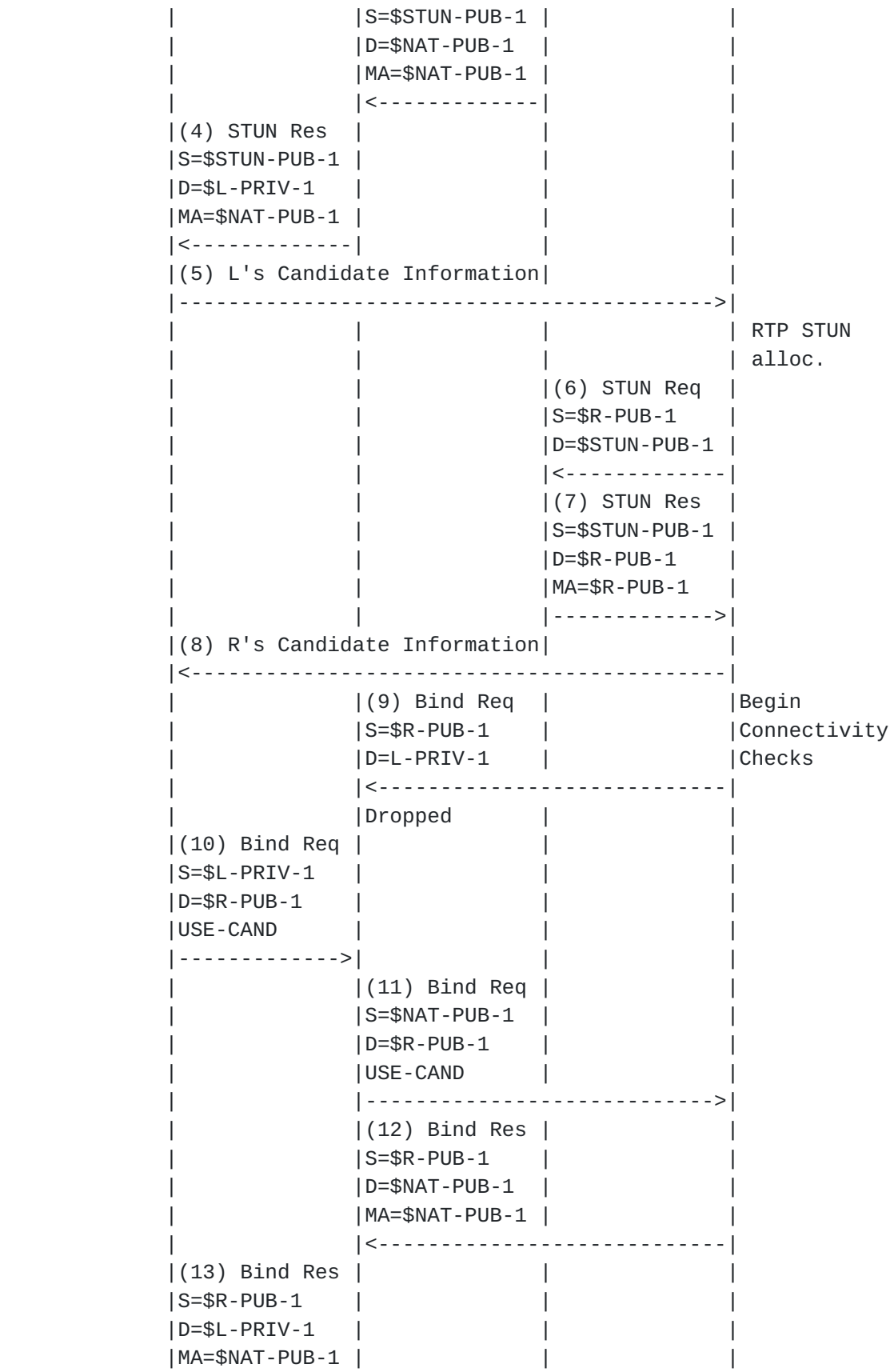
To facilitate understanding, transport addresses are listed using variables that have mnemonic names. The format of the name is entity-type-seqno, where entity refers to the entity whose IP address the transport address is on, and is one of "L", "R", "STUN", or "NAT". The type is either "PUB" for transport addresses that are public, and "PRIV" for transport addresses that are private. Finally, seq-no is a sequence number that is different for each transport address of the same type on a particular entity. Each variable has an IP address and port, denoted by varname.IP and varname.PORT, respectively, where varname is the name of the variable.

The STUN server has advertised transport address STUN-PUB-1 (which is 192.0.2.2:3478).

In the call flow itself, STUN messages are annotated with several attributes. The "S=" attribute indicates the source transport address of the message. The "D=" attribute indicates the destination transport address of the message. The "MA=" attribute is used in STUN Binding response messages and refers to the mapped address. "USE-CAND" implies the presence of the USE-CANDIDATE attribute.

The call flow examples omit STUN authentication operations and RTCP, and focus on RTP for a single media stream between two full implementations.




|S=$STUN-PUB-1 |
|D=$NAT-PUB-1 |
|MA=$NAT-PUB-1 |
|<-------------|
|(4) STUN Res |
|S=$STUN-PUB-1 |
|D=$L-PRIV-1 |
|MA=$NAT-PUB-1 |
|<-------------|
|(5) L's Candidate Information|
|-->|
| RTP STUN
| alloc.
|(6) STUN Req |
|S=$R-PUB-1 |
|D=$STUN-PUB-1 |
|<-------------|
|(7) STUN Res |
|S=$STUN-PUB-1 |
|D=$R-PUB-1 |
|MA=$R-PUB-1 |
|------------->|
|(8) R's Candidate Information|
|<--|
|(9) Bind Req |
|S=$R-PUB-1 |
|D=L-PRIV-1 |
|<-----------------------------|
|Dropped |
|(10) Bind Req |
|S=$L-PRIV-1 |
|D=$R-PUB-1 |
|USE-CAND |
|------------->|
|(11) Bind Req |
|S=$NAT-PUB-1 |
|D=$R-PUB-1 |
|USE-CAND |
|----------------------------->|
|(12) Bind Res |
|S=$R-PUB-1 |
|D=$NAT-PUB-1 |
|MA=$NAT-PUB-1 |
|<-----------------------------|
|(13) Bind Res |
|S=$R-PUB-1 |
|D=$L-PRIV-1 |
|MA=$NAT-PUB-1 |
Begin
Connectivity
Checks

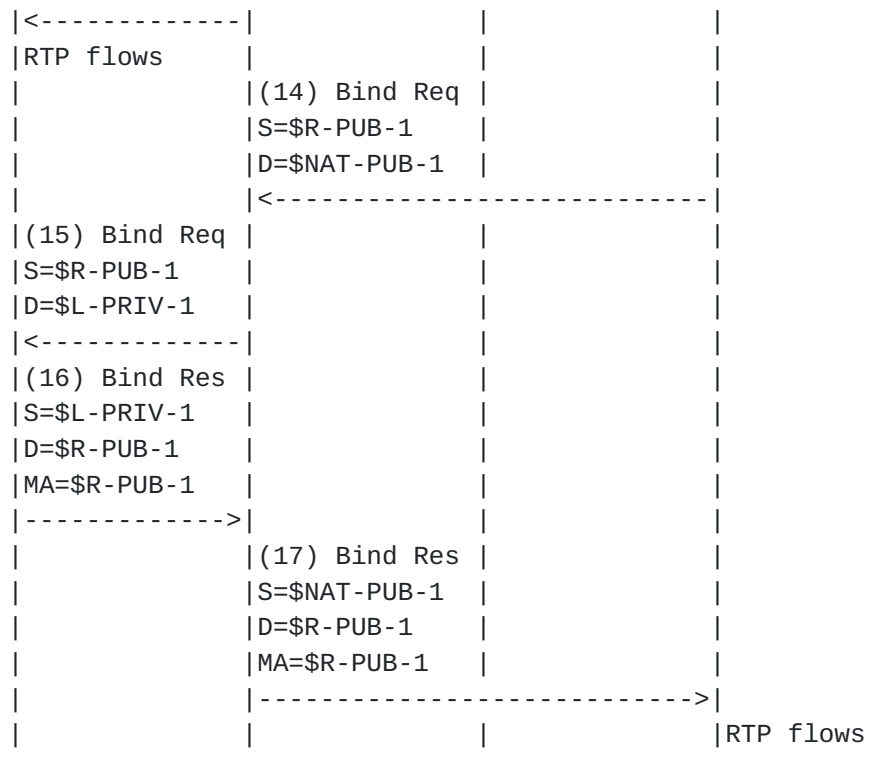


Figure 9: Example Flow

First, agent L obtains a host candidate from its local IP address (not shown), and from that, sends a STUN Binding request to the STUN server to get a server reflexive candidate (messages 1-4). Recall that the NAT has the address and port independent mapping property. Here, it creates a binding of NAT-PUB-1 for this UDP request, and this becomes the server reflexive candidate for RTP.

Agent L sets a type preference of 126 for the host candidate and 100 for the server reflexive. The local preference is 65535. Based on this, the priority of the host candidate is 2130706431 and for the server reflexive candidate is 1694498815. The host candidate is assigned a foundation of 1, and the server reflexive, a foundation of 2. These are sent to the peer.

This candidate information is received at agent R. Agent R will obtain a host candidate, and from it, obtain a server reflexive candidate (messages 6-7). Since R is not behind a NAT, this candidate is identical to its host candidate, and they share the same base. It therefore discards this redundant candidate and ends up with a single host candidate. With identical type and local preferences as L, the priority for this candidate is 2130706431. It chooses a foundation of 1 for its single candidate. Then R's candidates are then sent to L.

Since neither side indicated that it is lite, the initiating agent that began ICE processing (agent L) becomes the controlling agent.

Agents L and R both pair up the candidates. They both initially have two pairs. However, agent L will prune the pair containing its server reflexive candidate, resulting in just one. At agent L, this pair has a local candidate of $L_PRIV_1 and remote candidate of $R_PUB_1, and has a candidate pair priority of 4.57566E+18 (note that an implementation would represent this as a 64-bit integer so as not to lose precision). At agent R, there are two pairs. The highest priority has a local candidate of $R_PUB_1 and remote candidate of $L_PRIV_1 and has a priority of 4.57566E+18, and the second has a local candidate of $R_PUB_1 and remote candidate of $NAT_PUB_1 and priority 3.63891E+18.

Agent R begins its connectivity check (message 9) for the first pair (between the two host candidates). Since R is the controlled agent for this session, the check omits the USE-CANDIDATE attribute. The host candidate from agent L is private and behind a NAT, and thus this check won't be successful, because the packet cannot be routed from R to L.

When agent L gets the R's candidates, it performs its one and only connectivity check (messages 10-13). It implements the aggressive nomination algorithm, and thus includes a USE-CANDIDATE attribute in this check. Since the check succeeds, agent L creates a new pair, whose local candidate is from the mapped address in the Binding response (NAT-PUB-1 from message 13) and whose remote candidate is the destination of the request (R-PUB-1 from message 10). This is added to the valid list. In addition, it is marked as selected since the Binding request contained the USE-CANDIDATE attribute. Since there is a selected candidate in the Valid list for the one component of this media stream, ICE processing for this stream moves into the Completed state. Agent L can now send media if it so chooses.

Soon after receipt of the STUN Binding request from agent L (message 11), agent R will generate its triggered check. This check happens to match the next one on its check list -- from its host candidate to agent L's server reflexive candidate. This check (messages 14-17) will succeed. Consequently, agent R constructs a new candidate pair using the mapped address from the response as the local candidate (R-PUB-1) and the destination of the request (NAT-PUB-1) as the remote candidate. This pair is added to the Valid list for that media stream. Since the check was generated in the reverse direction of a check that contained the USE-CANDIDATE attribute, the candidate pair is marked as selected. Consequently, processing for this stream moves into the Completed state, and agent R can also send media.

15. Security Considerations

There are several types of attacks possible in an ICE system. This section considers these attacks and their countermeasures. These countermeasures include:

- Using ICE in conjunction with secure signaling techniques, such as SIPS.

- Limiting the total number of connectivity checks to 100, and optionally limiting the number of candidates they'll accept in an candidate exchange.

15.1. Attacks on Connectivity Checks

An attacker might attempt to disrupt the STUN connectivity checks. Ultimately, all of these attacks fool an agent into thinking something incorrect about the results of the connectivity checks. The possible false conclusions an attacker can try and cause are:

False Invalid: An attacker can fool a pair of agents into thinking a candidate pair is invalid, when it isn't. This can be used to cause an agent to prefer a different candidate (such as one injected by the attacker) or to disrupt a call by forcing all candidates to fail.

False Valid: An attacker can fool a pair of agents into thinking a candidate pair is valid, when it isn't. This can cause an agent to proceed with a session, but then not be able to receive any media.

False Peer Reflexive Candidate: An attacker can cause an agent to discover a new peer reflexive candidate, when it shouldn't have. This can be used to redirect media streams to a Denial-of-Service (DoS) target or to the attacker, for eavesdropping or other purposes.

False Valid on False Candidate: An attacker has already convinced an agent that there is a candidate with an address that doesn't actually route to that agent (for example, by injecting a false peer reflexive candidate or false server reflexive candidate). It must then launch an attack that forces the agents to believe that this candidate is valid.

If an attacker can cause a false peer reflexive candidate or false valid on a false candidate, it can launch any of the attacks described in [RFC5389].

To force the false invalid result, the attacker has to wait for the connectivity check from one of the agents to be sent. When it is, the attacker needs to inject a fake response with an unrecoverable error response, such as a 400. However, since the candidate is, in fact, valid, the original request may reach the peer agent, and result in a success response. The attacker needs to force this packet or its response to be dropped, through a DoS attack, layer 2 network disruption, or other technique. If it doesn't do this, the success response will also reach the originator, alerting it to a possible attack. Fortunately, this attack is mitigated completely through the STUN short-term credential mechanism. The attacker needs to inject a fake response, and in order for this response to be processed, the attacker needs the password. If the candidate exchange signaling is secured, the attacker will not have the password and its response will be discarded.

Forcing the fake valid result works in a similar way. The agent needs to wait for the Binding request from each agent, and inject a fake success response. The attacker won't need to worry about disrupting the actual response since, if the candidate is not valid, it presumably wouldn't be received anyway. However, like the fake invalid attack, this attack is mitigated by the STUN short-term credential mechanism in conjunction with a secure candidate exchange.

Forcing the false peer reflexive candidate result can be done either with fake requests or responses, or with replays. We consider the fake requests and responses case first. It requires the attacker to send a Binding request to one agent with a source IP address and port for the false candidate. In addition, the attacker must wait for a Binding request from the other agent, and generate a fake response with a XOR-MAPPED-ADDRESS attribute containing the false candidate. Like the other attacks described here, this attack is mitigated by the STUN message integrity mechanisms and secure candidate exchanges.

Forcing the false peer reflexive candidate result with packet replays is different. The attacker waits until one of the agents sends a check. It intercepts this request, and replays it towards the other agent with a faked source IP address. It must also prevent the original request from reaching the remote agent, either by launching a DoS attack to cause the packet to be dropped, or forcing it to be dropped using layer 2 mechanisms. The replayed packet is received at the other agent, and accepted, since the integrity check passes (the integrity check cannot and does not cover the source IP address and port). It is then responded to. This response will contain a XOR-MAPPED-ADDRESS with the false candidate, and will be sent to that false candidate. The attacker must then receive it and relay it towards the originator.

The other agent will then initiate a connectivity check towards that false candidate. This validation needs to succeed. This requires the attacker to force a false valid on a false candidate. Injecting of fake requests or responses to achieve this goal is prevented using the integrity mechanisms of STUN and the candidate exchange. Thus, this attack can only be launched through replays. To do that, the attacker must intercept the check towards this false candidate, and replay it towards the other agent. Then, it must intercept the response and replay that back as well.

This attack is very hard to launch unless the attacker is identified by the fake candidate. This is because it requires the attacker to intercept and replay packets sent by two different hosts. If both agents are on different networks (for example, across the public Internet), this attack can be hard to coordinate, since it needs to occur against two different endpoints on different parts of the network at the same time.

If the attacker itself is identified by the fake candidate, the attack is easier to coordinate. However, if the media path is secured (e.g., using SRTP [RFC3711]), the attacker will not be able to play the media packets, but will only be able to discard them, effectively disabling the media stream for the call. However, this attack requires the agent to disrupt packets in order to block the connectivity check from reaching the target. In that case, if the goal is to disrupt the media stream, it's much easier to just disrupt it with the same mechanism, rather than attack ICE.

15.2. Attacks on Server Reflexive Address Gathering

ICE endpoints make use of STUN Binding requests for gathering server reflexive candidates from a STUN server. These requests are not authenticated in any way. As a consequence, there are numerous techniques an attacker can employ to provide the client with a false server reflexive candidate:

- An attacker can compromise the DNS, causing DNS queries to return a rogue STUN server address. That server can provide the client with fake server reflexive candidates. This attack is mitigated by DNS security, though DNS-SEC is not required to address it.

- An attacker that can observe STUN messages (such as an attacker on a shared network segment, like WiFi) can inject a fake response that is valid and will be accepted by the client.

- An attacker can compromise a STUN server by means of a virus, and cause it to send responses with incorrect mapped addresses.

A false mapped address learned by these attacks will be used as a server reflexive candidate in the ICE exchange. For this candidate to actually be used for media, the attacker must also attack the connectivity checks, and in particular, force a false valid on a false candidate. This attack is very hard to launch if the false address identifies a fourth party (neither the initiator, responder, nor attacker), since it requires attacking the checks generated by each agent in the session, and is prevented by SRTP if it identifies the attacker themself.

If the attacker elects not to attack the connectivity checks, the worst it can do is prevent the server reflexive candidate from being used. However, if the peer agent has at least one candidate that is reachable by the agent under attack, the STUN connectivity checks themselves will provide a peer reflexive candidate that can be used for the exchange of media. Peer reflexive candidates are generally preferred over server reflexive candidates. As such, an attack solely on the STUN address gathering will normally have no impact on a session at all.

15.3. Attacks on Relayed Candidate Gathering

An attacker might attempt to disrupt the gathering of relayed candidates, forcing the client to believe it has a false relayed candidate. Exchanges with the TURN server are authenticated using a long-term credential. Consequently, injection of fake responses or requests will not work. In addition, unlike Binding requests, Allocate requests are not susceptible to replay attacks with modified source IP addresses and ports, since the source IP address and port are not utilized to provide the client with its relayed candidate.

However, TURN servers are susceptible to DNS attacks, or to viruses aimed at the TURN server, for purposes of turning it into a zombie or rogue server. These attacks can be mitigated by DNS-SEC and through good box and software security on TURN servers.

Even if an attacker has caused the client to believe in a false relayed candidate, the connectivity checks cause such a candidate to be used only if they succeed. Thus, an attacker must launch a false valid on a false candidate, per above, which is a very difficult attack to coordinate.

15.4. Insider Attacks

In addition to attacks where the attacker is a third party trying to insert fake candidate information or stun messages, there are attacks possible with ICE when the attacker is an authenticated and valid participant in the ICE exchange.

15.4.1. STUN Amplification Attack

The STUN amplification attack is similar to the voice hammer. However, instead of voice packets being directed to the target, STUN connectivity checks are directed to the target. The attacker sends an a large number of candidates, say, 50. The responding agent receives the candidate information, and starts its checks, which are directed at the target, and consequently, never generate a response. The answerer will start a new connectivity check every Ta ms (say, Ta=20ms). However, the retransmission timers are set to a large number due to the large number of candidates. As a consequence, packets will be sent at an interval of one every Ta milliseconds, and then with increasing intervals after that. Thus, STUN will not send packets at a rate faster than media would be sent, and the STUN packets persist only briefly, until ICE fails for the session. Nonetheless, this is an amplification mechanism.

It is impossible to eliminate the amplification, but the volume can be reduced through a variety of heuristics. Agents SHOULD limit the total number of connectivity checks they perform to 100. Additionally, agents MAY limit the number of candidates they'll accept.

Frequently, protocols that wish to avoid these kinds of attacks force the initiator to wait for a response prior to sending the next message. However, in the case of ICE, this is not possible. It is not possible to differentiate the following two cases:

- There was no response because the initiator is being used to launch a DoS attack against an unsuspecting target that will not respond.

- There was no response because the IP address and port are not reachable by the initiator.

In the second case, another check should be sent at the next opportunity, while in the former case, no further checks should be sent.

16. STUN Extensions

16.1. New Attributes

This specification defines four new attributes, PRIORITY, USE-CANDIDATE, ICE-CONTROLLED, and ICE-CONTROLLING.

The PRIORITY attribute indicates the priority that is to be associated with a peer reflexive candidate, should one be discovered

by this check. It is a 32-bit unsigned integer, and has an attribute value of 0x0024.

The USE-CANDIDATE attribute indicates that the candidate pair resulting from this check should be used for transmission of media. The attribute has no content (the Length field of the attribute is zero); it serves as a flag. It has an attribute value of 0x0025.

The ICE-CONTROLLED attribute is present in a Binding request and indicates that the client believes it is currently in the controlled role. The content of the attribute is a 64-bit unsigned integer in network byte order, which contains a random number used for tie-breaking of role conflicts.

The ICE-CONTROLLING attribute is present in a Binding request and indicates that the client believes it is currently in the controlling role. The content of the attribute is a 64-bit unsigned integer in network byte order, which contains a random number used for tie-breaking of role conflicts.

16.2. New Error Response Codes

This specification defines a single error response code:

487 (Role Conflict): The Binding request contained either the ICE-CONTROLLING or ICE-CONTROLLED attribute, indicating a role that conflicted with the server. The server ran a tie-breaker based on the tie-breaker value in the request and determined that the client needs to switch roles.

17. Operational Considerations

This section discusses issues relevant to network operators looking to deploy ICE.

17.1. NAT and Firewall Types

ICE was designed to work with existing NAT and firewall equipment. Consequently, it is not necessary to replace or reconfigure existing firewall and NAT equipment in order to facilitate deployment of ICE. Indeed, ICE was developed to be deployed in environments where the Voice over IP (VoIP) operator has no control over the IP network infrastructure, including firewalls and NAT.

That said, ICE works best in environments where the NAT devices are "behave" compliant, meeting the recommendations defined in [RFC4787] and [RFC5382]. In networks with behave-compliant NAT, ICE will work without the need for a TURN server, thus improving voice quality,

decreasing call setup times, and reducing the bandwidth demands on the network operator.

17.2. Bandwidth Requirements

Deployment of ICE can have several interactions with available network capacity that operators should take into consideration.

17.2.1. STUN and TURN Server Capacity Planning

First and foremost, ICE makes use of TURN and STUN servers, which would typically be located in the network operator's data centers. The STUN servers require relatively little bandwidth. For each component of each media stream, there will be one or more STUN transactions from each client to the STUN server. In a basic voice-only IPv4 VoIP deployment, there will be four transactions per call (one for RTP and one for RTCP, for both caller and callee). Each transaction is a single request and a single response, the former being 20 bytes long, and the latter, 28. Consequently, if a system has N users, and each makes four calls in a busy hour, this would require N*1.7bps. For one million users, this is 1.7 Mbps, a very small number (relatively speaking).

TURN traffic is more substantial. The TURN server will see traffic volume equal to the STUN volume (indeed, if TURN servers are deployed, there is no need for a separate STUN server), in addition to the traffic for the actual media traffic. The amount of calls requiring TURN for media relay is highly dependent on network topologies, and can and will vary over time. In a network with 100% behave-compliant NAT, it is exactly zero. At time of writing, large-scale consumer deployments were seeing between 5 and 10 percent of calls requiring TURN servers. Considering a voice-only deployment using G.711 (so 80 kbps in each direction), with .2 erlangs during the busy hour, this is N*3.2 kbps. For a population of one million users, this is 3.2 Gbps, assuming a 10% usage of TURN servers.

17.2.2. Gathering and Connectivity Checks

The process of gathering of candidates and performing of connectivity checks can be bandwidth intensive. ICE has been designed to pace both of these processes. The gathering phase and the connectivity check phase are meant to generate traffic at roughly the same bandwidth as the media traffic itself. This was done to ensure that, if a network is designed to support multimedia traffic of a certain type (voice, video, or just text), it will have sufficient capacity to support the ICE checks for that media. Of course, the ICE checks will cause a marginal increase in the total utilization; however, this will typically be an extremely small increase.

Congestion due to the gathering and check phases has proven to be a problem in deployments that did not utilize pacing. Typically, access links became congested as the endpoints flooded the network with checks as fast as they can send them. Consequently, network operators should make sure that their ICE implementations support the pacing feature. Though this pacing does increase call setup times, it makes ICE network friendly and easier to deploy.

17.2.3. Keepalives

STUN keepalives (in the form of STUN Binding Indications) are sent in the middle of a media session. However, they are sent only in the absence of actual media traffic. In deployments that are not utilizing Voice Activity Detection (VAD), the keepalives are never used and there is no increase in bandwidth usage. When VAD is being used, keepalives will be sent during silence periods. This involves a single packet every 15-20 seconds, far less than the packet every 20-30 ms that is sent when there is voice. Therefore, keepalives don't have any real impact on capacity planning.

17.3. ICE and ICE-lite

Deployments utilizing a mix of ICE and ICE-lite interoperate perfectly. They have been explicitly designed to do so, without loss of function.

However, ICE-lite can only be deployed in limited use cases. Those cases, and the caveats involved in doing so, are documented in Appendix A.

17.4. Troubleshooting and Performance Management

ICE utilizes end-to-end connectivity checks, and places much of the processing in the endpoints. This introduces a challenge to the network operator -- how can they troubleshoot ICE deployments? How can they know how ICE is performing?

ICE has built-in features to help deal with these problems. SIP servers on the signaling path, typically deployed in the data centers of the network operator, will see the contents of the candidate exchanges that convey the ICE parameters. These parameters include the type of each candidate (host, server reflexive, or relayed), along with their related addresses. Once ICE processing has completed, an updated candidate exchange takes place, signaling the selected address (and its type). This updated re-INVITE is performed exactly for the purposes of educating network equipment (such as a diagnostic tool attached to a SIP server) about the results of ICE processing.

As a consequence, through the logs generated by the SIP server, a network operator can observe what types of candidates are being used for each call, and what address was selected by ICE. This is the primary information that helps evaluate how ICE is performing.

17.5. Endpoint Configuration

ICE relies on several pieces of data being configured into the endpoints. This configuration data includes timers, credentials for TURN servers, and hostnames for STUN and TURN servers. ICE itself does not provide a mechanism for this configuration. Instead, it is assumed that this information is attached to whatever mechanism is used to configure all of the other parameters in the endpoint. For SIP phones, standard solutions such as the configuration framework [RFC6080] have been defined.

18. IANA Considerations

The original ICE specification registered four new STUN attributes, and one new STUN error response. The STUN attributes and error response are reproduced here. In addition, this specification registers a new ICE option.

18.1. STUN Attributes

IANA has registered four STUN attributes:

0x0024 PRIORITY
0x0025 USE-CANDIDATE
0x8029 ICE-CONTROLLED
0x802A ICE-CONTROLLING

18.2. STUN Error Responses

IANA has registered following STUN error response code:

487 Role Conflict: The client asserted an ICE role (controlling or controlled) that is in conflict with the role of the server.

18.3. ICE Options

IANA is requested to register the following ICE option in the "ICE Options" sub-registry of the "Interactive Connectivity Establishment (ICE) registry", following the procedures defined in [RFC6336].

ICE Option name:

ice2

Contact:

Name: Christer Holmberg
E-mail: christer.holmberg(at)ericsson(dot)com
Address: Oy LM Ericsson Ab, 02420 Jorvas, FINLAND

Change control:

IESG

Description:

The ICE option indicates that the ICE agent using the ICE option is compliant and implemented according to RFC XXXX.

Reference:

RFC XXXX

19. IAB Considerations

The IAB has studied the problem of "Unilateral Self-Address Fixing", which is the general process by which a agent attempts to determine its address in another realm on the other side of a NAT through a collaborative protocol reflection mechanism [RFC3424]. 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. Indeed, ICE can be considered a B-SAF (Bilateral Self-Address Fixing) protocol, rather than an UNSAF protocol. Regardless, the IAB has mandated that any protocols developed for this purpose document a specific set of considerations. This section meets those requirements.

19.1. 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 that can be used for communication.
>
> Provide a means for a agent to determine an address that is reachable by another peer with which it wishes to communicate.

19.2. 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. ICE also helps prevent certain security attacks that have nothing to do with NAT. 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, server reflexive and relayed candidates (both forms of UNSAF addresses) simply never get used, because higher-priority connectivity exists to the native host candidates. Therefore, the servers get used less and less, and can eventually be remove when their usage goes to zero.

Indeed, ICE can assist in the transition from IPv4 to IPv6. It can be used to determine whether to use IPv6 or IPv4 when two dual-stack hosts communicate with SIP (IPv6 gets used). It can also allow a network with both 6to4 and native v6 connectivity to determine which address to use when communicating with a peer.

19.3. Brittleness Introduced by ICE

>From RFC 3424, 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, classic STUN (as described in RFC 3489 [RFC3489]) has several points of brittleness. One of them is the discovery process

that requires an agent to try to 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.

Another point of brittleness in classic STUN 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 agents to directly connect if possible. Therefore, in some cases, the failure of a STUN server would still allow for a call to progress when ICE is used.

Another point of brittleness in classic 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 classic STUN is that it doesn't work in all network topologies. In cases where there is a shared NAT between each agent and the STUN server, traditional STUN may not work. With ICE, that restriction is removed.

Classic STUN also introduces some security considerations. Fortunately, those security considerations are also mitigated by ICE.

Consequently, ICE serves to repair the brittleness introduced in classic STUN, and does not introduce any additional brittleness into the system.

The penalty of these improvements is that ICE increases session establishment times.

19.4. Requirements for a Long-Term Solution

From RFC 3424, any UNSAF proposal must provide:

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

Our conclusions from RFC 3489 remain unchanged. However, we feel ICE actually helps because we believe it can be part of the long-term solution.

19.5. Issues with Existing NAPT Boxes

From RFC 3424, any UNSAF proposal must provide:

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

A number of NAT boxes are now being deployed into the market that try to 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 interferes with classic STUN. However, the update to STUN [RFC5389] uses an encoding that hides these binary addresses from generic ALGs.

Existing NAPT boxes have non-deterministic and typically short expiration times for UDP-based bindings. This requires implementations to send periodic keepalives to maintain those bindings. ICE uses a default of 15 s, which is a very conservative estimate. Eventually, over time, as NAT boxes become compliant to behave [RFC4787], this minimum keepalive will become deterministic and well-known, and the ICE timers can be adjusted. Having a way to discover and control the minimum keepalive interval would be far better still.

20. Changes from RFC 5245

Following is the list of changes from RFC 5245

- The specification was generalized to be more usable with any protocol and the parts that are specific to SIP and SDP were moved to a SIP/SDP usage document [I-D.ietf-mmusic-ice-sip-sdp].

- Default candidates, multiple components, ICE mismatch detection, subsequent offer/answer, and role conflict resolution were made optional since they are not needed with every protocol using ICE.

- With IPv6, the precedence rules of RFC 6724 are used instead of the obsoleted RFC 3483 and using address preferences provided by the host operating system is recommended.

- Candidate gathering rules regarding loopback addresses and IPv6 addresses were clarified.

21. Acknowledgements

Most of the text in this document comes from the original ICE specification, RFC 5245. The authors would like to thank everyone who has contributed to that document. For additional contributions

to this revision of the specification we would like to thank Emil Ivov, Paul Kyzivat, Pal-Erik Martinsen, Simon Perrault, Eric Rescorla, Thomas Stach, Peter Thatcher, Martin Thomson, Justin Uberti, and Suhas Nandakumar.

22. References

22.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <http://www.rfc-editor.org/info/rfc5389>.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, DOI 10.17487/RFC5766, April 2010, <http://www.rfc-editor.org/info/rfc5766>.

[RFC6336] Westerlund, M. and C. Perkins, "IANA Registry for Interactive Connectivity Establishment (ICE) Options", RFC 6336, DOI 10.17487/RFC6336, July 2011, <http://www.rfc-editor.org/info/rfc6336>.

[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <http://www.rfc-editor.org/info/rfc6724>.

22.2. Informative References

[RFC3605] Huitema, C., "Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)", RFC 3605, DOI 10.17487/RFC3605, October 2003, <http://www.rfc-editor.org/info/rfc3605>.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <http://www.rfc-editor.org/info/rfc3261>.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <http://www.rfc-editor.org/info/rfc3264>.

[RFC3489] Rosenberg, J., Weinberger, J., Huitema, C., and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", RFC 3489, DOI 10.17487/RFC3489, March 2003, <http://www.rfc-editor.org/info/rfc3489>.

[RFC3235] Senie, D., "Network Address Translator (NAT)-Friendly Application Design Guidelines", RFC 3235, DOI 10.17487/RFC3235, January 2002, <http://www.rfc-editor.org/info/rfc3235>.

[RFC3303] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A., and A. Rayhan, "Middlebox communication architecture and framework", RFC 3303, DOI 10.17487/RFC3303, August 2002, <http://www.rfc-editor.org/info/rfc3303>.

[RFC3102] Borella, M., Lo, J., Grabelsky, D., and G. Montenegro, "Realm Specific IP: Framework", RFC 3102, DOI 10.17487/RFC3102, October 2001, <http://www.rfc-editor.org/info/rfc3102>.

[RFC3103] Borella, M., Grabelsky, D., Lo, J., and K. Taniguchi, "Realm Specific IP: Protocol Specification", RFC 3103, DOI 10.17487/RFC3103, October 2001, <http://www.rfc-editor.org/info/rfc3103>.

[RFC3424] Daigle, L., Ed. and IAB, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation", RFC 3424, DOI 10.17487/RFC3424, November 2002, <http://www.rfc-editor.org/info/rfc3424>.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <http://www.rfc-editor.org/info/rfc3550>.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <http://www.rfc-editor.org/info/rfc3711>.

[RFC3056] Carpenter, B. and K. Moore, "Connection of IPv6 Domains via IPv4 Clouds", RFC 3056, DOI 10.17487/RFC3056, February 2001, <http://www.rfc-editor.org/info/rfc3056>.

[RFC3389] Zopf, R., "Real-time Transport Protocol (RTP) Payload for Comfort Noise (CN)", RFC 3389, DOI 10.17487/RFC3389, September 2002, <http://www.rfc-editor.org/info/rfc3389>.

[RFC3879] Huitema, C. and B. Carpenter, "Deprecating Site Local Addresses", RFC 3879, DOI 10.17487/RFC3879, September 2004, <http://www.rfc-editor.org/info/rfc3879>.

[RFC4038] Shin, M-K., Ed., Hong, Y-G., Hagino, J., Savola, P., and E. Castro, "Application Aspects of IPv6 Transition", RFC 4038, DOI 10.17487/RFC4038, March 2005, <http://www.rfc-editor.org/info/rfc4038>.

[RFC4091] Camarillo, G. and J. Rosenberg, "The Alternative Network Address Types (ANAT) Semantics for the Session Description Protocol (SDP) Grouping Framework", RFC 4091, DOI 10.17487/RFC4091, June 2005, <http://www.rfc-editor.org/info/rfc4091>.

[RFC4092] Camarillo, G. and J. Rosenberg, "Usage of the Session Description Protocol (SDP) Alternative Network Address Types (ANAT) Semantics in the Session Initiation Protocol (SIP)", RFC 4092, DOI 10.17487/RFC4092, June 2005, <http://www.rfc-editor.org/info/rfc4092>.

[RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 4291, DOI 10.17487/RFC4291, February 2006, <http://www.rfc-editor.org/info/rfc4291>.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <http://www.rfc-editor.org/info/rfc4566>.

[RFC2475] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and W. Weiss, "An Architecture for Differentiated Services", RFC 2475, DOI 10.17487/RFC2475, December 1998, <http://www.rfc-editor.org/info/rfc2475>.

[RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <http://www.rfc-editor.org/info/rfc1918>.

[RFC4787] Audet, F., Ed. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, DOI 10.17487/RFC4787, January 2007, <http://www.rfc-editor.org/info/rfc4787>.

[I-D.ietf-avt-rtp-no-op] Andreasen, F., "A No-Op Payload Format for RTP", draft-ietf-avt-rtp-no-op-04 (work in progress), May 2007.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, DOI 10.17487/RFC5761, April 2010, <http://www.rfc-editor.org/info/rfc5761>.

[RFC4103] Hellstrom, G. and P. Jones, "RTP Payload for Text Conversation", RFC 4103, DOI 10.17487/RFC4103, June 2005, <http://www.rfc-editor.org/info/rfc4103>.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <http://www.rfc-editor.org/info/rfc5245>.

[RFC5382] Guha, S., Ed., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, DOI 10.17487/RFC5382, October 2008, <http://www.rfc-editor.org/info/rfc5382>.

[RFC6080] Petrie, D. and S. Channabasappa, Ed., "A Framework for Session Initiation Protocol User Agent Profile Delivery", RFC 6080, DOI 10.17487/RFC6080, March 2011, <http://www.rfc-editor.org/info/rfc6080>.

[RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <http://www.rfc-editor.org/info/rfc6146>.

[RFC6147] Bagnulo, M., Sullivan, A., Matthews, P., and I. van Beijnum, "DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers", RFC 6147, DOI 10.17487/RFC6147, April 2011, <http://www.rfc-editor.org/info/rfc6147>.

[RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, DOI 10.17487/RFC6544, March 2012, <http://www.rfc-editor.org/info/rfc6544>.

[RFC7050] Savolainen, T., Korhonen, J., and D. Wing, "Discovery of the IPv6 Prefix Used for IPv6 Address Synthesis", RFC 7050, DOI 10.17487/RFC7050, November 2013, <http://www.rfc-editor.org/info/rfc7050>.

[I-D.ietf-mmusic-ice-sip-sdp] Petit-Huguenin, M., Keranen, A., and S. Nandakumar, "Using Interactive Connectivity Establishment (ICE) with Session Description Protocol (SDP) offer/answer and Session Initiation Protocol (SIP)", draft-ietf-mmusic-ice-sip-sdp-08 (work in progress), March 2016.

[I-D.ietf-6man-ipv6-address-generation-privacy] Cooper, A., Gont, F., and D. Thaler, "Privacy Considerations for IPv6 Address Generation Mechanisms", draft-ietf-6man-ipv6-address-generation-privacy-08 (work in progress), September 2015.

Appendix A. Lite and Full Implementations

ICE allows for two types of implementations. A full implementation supports the controlling and controlled roles in a session, and can also perform address gathering. In contrast, a lite implementation is a minimalist implementation that does little but respond to STUN checks.

Because ICE requires both endpoints to support it in order to bring benefits to either endpoint, incremental deployment of ICE in a network is more complicated. Many sessions involve an endpoint that is, by itself, not behind a NAT and not one that would worry about NAT traversal. A very common case is to have one endpoint that requires NAT traversal (such as a VoIP hard phone or soft phone) make a call to one of these devices. Even if the phone supports a full ICE implementation, ICE won't be used at all if the other device doesn't support it. The lite implementation allows for a low-cost entry point for these devices. Once they support the lite implementation, full implementations can connect to them and get the full benefits of ICE.

Consequently, a lite implementation is only appropriate for devices that will *always* be connected to the public Internet and have a public IP address at which it can receive packets from any

correspondent. ICE will not function when a lite implementation is placed behind a NAT.

ICE allows a lite implementation to have a single IPv4 host candidate and several IPv6 addresses. In that case, candidate pairs are selected by the controlling agent using a static algorithm, such as the one in RFC 6724, which is recommended by this specification. However, static mechanisms for address selection are always prone to error, since they cannot ever reflect the actual topology and can never provide actual guarantees on connectivity. They are always heuristics. Consequently, if an agent is implementing ICE just to select between its IPv4 and IPv6 addresses, and none of its IP addresses are behind NAT, usage of full ICE is still RECOMMENDED in order to provide the most robust form of address selection possible.

It is important to note that the lite implementation was added to this specification to provide a stepping stone to full implementation. Even for devices that are always connected to the public Internet with just a single IPv4 address, a full implementation is preferable if achievable. Full implementations also obtain the security benefits of ICE unrelated to NAT traversal; in particular, the voice hammer attack described in Section 15 is prevented only for full implementations, not lite. Finally, it is often the case that a device that finds itself with a public address today will be placed in a network tomorrow where it will be behind a NAT. It is difficult to definitively know, over the lifetime of a device or product, that it will always be used on the public Internet. Full implementation provides assurance that communications will always work.

Appendix B. Design Motivations

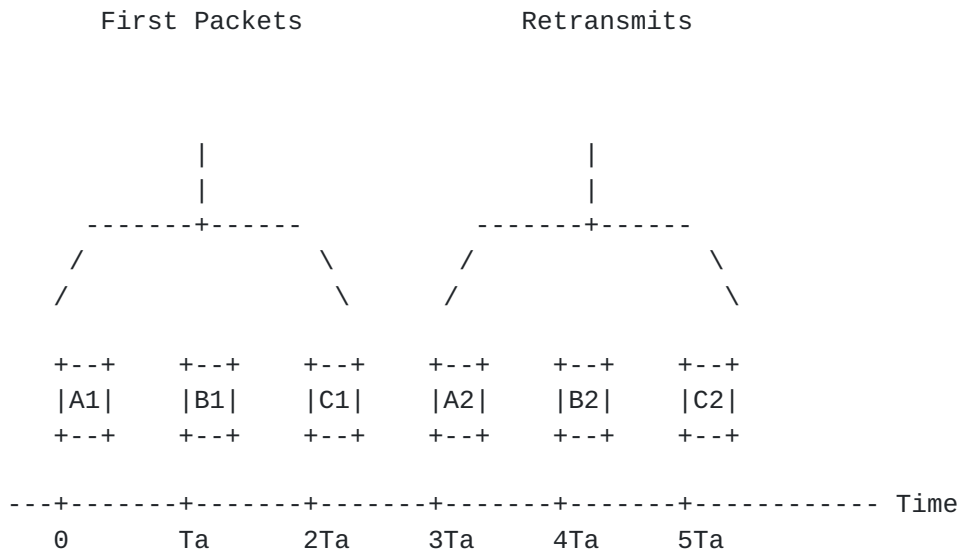
ICE contains a number of normative behaviors that may themselves be simple, but derive from complicated or non-obvious thinking or use cases that merit further discussion. Since these design motivations are not necessary to understand for purposes of implementation, they are discussed here in an appendix to the specification. This section is non-normative.

B.1. Pacing of STUN Transactions

STUN transactions used to gather candidates and to verify connectivity are paced out at an approximate rate of one new transaction every Ta milliseconds. Each transaction, in turn, has a retransmission timer RTO that is a function of Ta as well. Why are these transactions paced, and why are these formulas used?

Sending of these STUN requests will often have the effect of creating bindings on NAT devices between the client and the STUN servers. Experience has shown that many NAT devices have upper limits on the rate at which they will create new bindings. Experiments have shown that once every 20 ms is well supported, but not much lower than that. This is why Ta has a lower bound of 20 ms. Furthermore, transmission of these packets on the network makes use of bandwidth and needs to be rate limited by the agent. Deployments based on earlier draft versions of [RFC5245] tended to overload rate-constrained access links and perform poorly overall, in addition to negatively impacting the network. As a consequence, the pacing ensures that the NAT device does not get overloaded and that traffic is kept at a reasonable rate.

The definition of a "reasonable" rate is that STUN should not use more bandwidth than the RTP itself will use, once media starts flowing. The formula for Ta is designed so that, if a STUN packet were sent every Ta seconds, it would consume the same amount of bandwidth as RTP packets, summed across all media streams. Of course, STUN has retransmits, and the desire is to pace those as well. For this reason, RTO is set such that the first retransmit on the first transaction happens just as the first STUN request on the last transaction occurs. Pictorially:



In this picture, there are three transactions that will be sent (for example, in the case of candidate gathering, there are three host candidate/STUN server pairs). These are transactions A, B, and C. The retransmit timer is set so that the first retransmission on the first transaction (packet A2) is sent at time 3Ta.

Subsequent retransmits after the first will occur even less frequently than Ta milliseconds apart, since STUN uses an exponential back-off on its retransmissions.

B.2. Candidates with Multiple Bases

Section 4.1.3 talks about eliminating candidates that have the same transport address and base. However, candidates with the same transport addresses but different bases are not redundant. When can an agent have two candidates that have the same IP address and port, but different bases? Consider the topology of Figure 10:

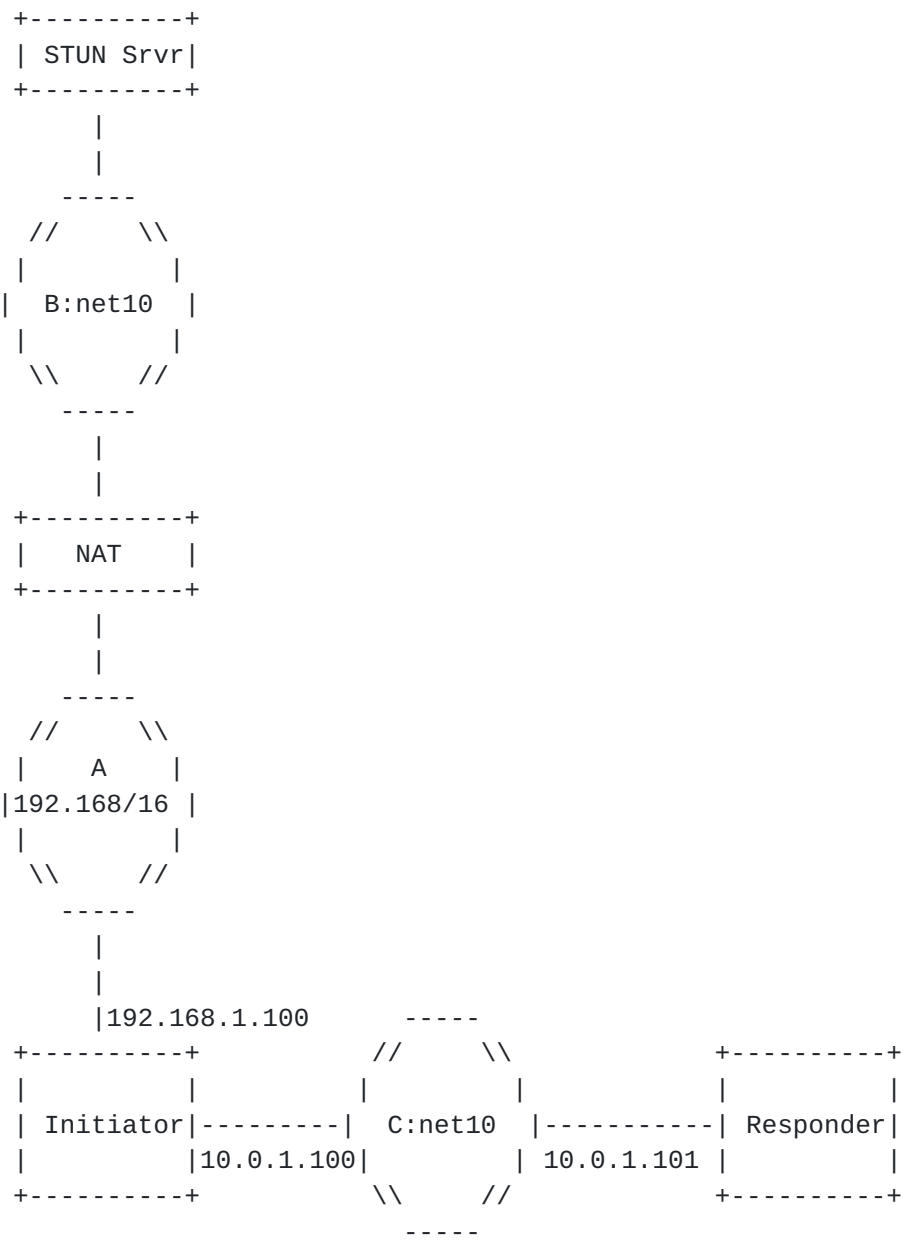


Figure 10: Identical Candidates with Different Bases

In this case, the initiating agent is multihomed. It has one IP address, 10.0.1.100, on network C, which is a net 10 private network. The responding agent is on this same network. The initiating agent is also connected to network A, which is 192.168/16 and has an IP address of 192.168.1.100 on this network. There is a NAT on this network, natting into network B, which is another net 10 private network, but not connected to network C. There is a STUN server on network B.

The initiating agent obtains a host candidate on its IP address on network C (10.0.1.100:2498) and a host candidate on its IP address on network A (192.168.1.100:3344). It performs a STUN query to its configured STUN server from 192.168.1.100:3344. This query passes through the NAT, which happens to assign the binding 10.0.1.100:2498. The STUN server reflects this in the STUN Binding response. Now, the initiating agent has obtained a server reflexive candidate with a transport address that is identical to a host candidate (10.0.1.100:2498). However, the server reflexive candidate has a base of 192.168.1.100:3344, and the host candidate has a base of 10.0.1.100:2498.

B.3. Purpose of the Related Address and Related Port Attributes

The candidate attribute contains two values that are not used at all by ICE itself -- related address and related port. Why are they present?

There are two motivations for its inclusion. The first is diagnostic. It is very useful to know the relationship between the different types of candidates. By including it, an agent can know which relayed candidate is associated with which reflexive candidate, which in turn is associated with a specific host candidate. When checks for one candidate succeed and not for others, this provides useful diagnostics on what is going on in the network.

The second reason has to do with off-path Quality of Service (QoS) mechanisms. When ICE is used in environments such as PacketCable 2.0, proxies will, in addition to performing normal SIP operations, inspect the SDP in SIP messages, and extract the IP address and port for media traffic. They can then interact, through policy servers, with access routers in the network, to establish guaranteed QoS for the media flows. This QoS is provided by classifying the RTP traffic based on 5-tuple, and then providing it a guaranteed rate, or marking its Diffserv codepoints appropriately. When a residential NAT is present, and a relayed candidate gets selected for media, this relayed candidate will be a transport address on an actual TURN server. That address says nothing about the actual transport address in the access router that would be used to classify packets for QoS treatment. Rather, the server reflexive candidate towards the TURN server is needed. By carrying the translation in the SDP, the proxy can use that transport address to request QoS from the access router.

B.4. Importance of the STUN Username

ICE requires the usage of message integrity with STUN using its short-term credential functionality. The actual short-term credential is formed by exchanging username fragments in the

candidate exchange. The need for this mechanism goes beyond just security; it is actually required for correct operation of ICE in the first place.

Consider agents L, R, and Z. L and R are within private enterprise 1, which is using 10.0.0.0/8. Z is within private enterprise 2, which is also using 10.0.0.0/8. As it turns out, R and Z both have IP address 10.0.1.1. L sends candidates to Z. Z, in responds L with its host candidates. In this case, those candidates are 10.0.1.1:8866 and 10.0.1.1:8877. As it turns out, R is in a session at that same time, and is also using 10.0.1.1:8866 and 10.0.1.1:8877 as host candidates. This means that R is prepared to accept STUN messages on those ports, just as Z is. L will send a STUN request to 10.0.1.1:8866 and another to 10.0.1.1:8877. However, these do not go to Z as expected. Instead, they go to R! If R just replied to them, L would believe it has connectivity to Z, when in fact it has connectivity to a completely different user, R. To fix this, the STUN short-term credential mechanisms are used. The username fragments are sufficiently random that it is highly unlikely that R would be using the same values as Z. Consequently, R would reject the STUN request since the credentials were invalid. In essence, the STUN username fragments provide a form of transient host identifiers, bound to a particular session established as part of the candidate exchange.

An unfortunate consequence of the non-uniqueness of IP addresses is that, in the above example, R might not even be an ICE agent. 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 affected. Fortunately, since the ports exchanged 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 R, but rather is the agent side of some protocol. This decreases the probability of hitting an allocated port, 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. Note that this is not a problem specific to ICE; stray packets can arrive at a port at any time for any type of protocol, especially ones on the public Internet. As such, this requirement is just restating a general design guideline for Internet applications -- be prepared for unknown packets on any port.

B.5. The Candidate Pair Priority Formula

The priority for a candidate pair has an odd form. It is:

pair priority = 2^32*MIN(G,D) + 2*MAX(G,D) + (G>D?1:0)

Why is this? When the candidate pairs are sorted based on this value, the resulting sorting has the MAX/MIN property. This means that the pairs are first sorted based on decreasing value of the minimum of the two priorities. For pairs that have the same value of the minimum priority, the maximum priority is used to sort amongst them. If the max and the min priorities are the same, the controlling agent's priority is used as the tie-breaker in the last part of the expression. The factor of 2*32 is used since the priority of a single candidate is always less than 2*32, resulting in the pair priority being a "concatenation" of the two component priorities. This creates the MAX/MIN sorting. MAX/MIN ensures that, for a particular agent, a lower-priority candidate is never used until all higher-priority candidates have been tried.

B.6. Why Are Keepalives Needed?

Once media begins flowing on a candidate pair, it is still necessary to keep the bindings alive at intermediate NATs for the duration of the session. Normally, the media stream packets themselves (e.g., RTP) meet this objective. 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 [RFC3264]. 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.

Secondly, some RTP payload formats, such as the payload format for text conversation [RFC4103], may send packets so infrequently that the interval exceeds the NAT binding timeouts.

Thirdly, if silence suppression is in use, long periods of silence may cause media transmission to cease sufficiently long for NAT bindings to time out.

For these reasons, the media packets themselves cannot be relied upon. ICE defines a simple periodic keepalive utilizing STUN Binding indications. This makes its bandwidth requirements highly predictable, and thus amenable to QoS reservations.

B.7. Why Prefer Peer Reflexive Candidates?

Section 4.1.2 describes procedures for computing the priority of candidate based on its type and local preferences. That section requires that the type preference for peer reflexive candidates always be higher than server reflexive. Why is that? The reason has to do with the security considerations in Section 15. It is much easier for an attacker to cause an agent to use a false server reflexive candidate than it is for an attacker to cause an agent to use a false peer reflexive candidate. Consequently, attacks against address gathering with Binding requests are thwarted by ICE by preferring the peer reflexive candidates.

B.8. Why Are Binding Indications Used for Keepalives?

Media keepalives are described in Section 10. These keepalives make use of STUN when both endpoints are ICE capable. However, rather than using a Binding request transaction (which generates a response), the keepalives use an Indication. Why is that?

The primary reason has to do with network QoS mechanisms. Once media begins flowing, network elements will assume that the media stream has a fairly regular structure, making use of periodic packets at fixed intervals, with the possibility of jitter. If an agent is sending media packets, and then receives a Binding request, it would need to generate a response packet along with its media packets. This will increase the actual bandwidth requirements for the 5-tuple carrying the media packets, and introduce jitter in the delivery of those packets. Analysis has shown that this is a concern in certain layer 2 access networks that use fairly tight packet schedulers for media.

Additionally, using a Binding Indication allows integrity to be disabled, allowing for better performance. This is useful for large-scale endpoints, such as PSTN gateways and SBCs.

Authors' Addresses

Ari Keranen
Ericsson
Hirsalantie 11
02420 Jorvas
Finland

Email: ari.keranen@ericsson.com

Christer Holmberg
Ericsson
Hirsalantie 11
02420 Jorvas
Finland

Email: christer.holmberg@ericsson.com

Jonathan Rosenberg
jdrosen.net
Monmouth, NJ
US

Email: jdrosen@jdrosen.net
URI: http://www.jdrosen.net