ICE
Internet-Draft
Obsoletes: 5245 (if approved)
Intended status: Standards Track
Expires: May 7, 2018

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

November 3, 2017

Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal draft-ietf-ice-rfc5245bis-15

Abstract

This document describes a protocol for Network Address Translator (NAT) traversal for UDP-based communication. 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 May 7, 2018.

Copyright Notice

Copyright (c) 2017 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 Candidates 8
2.2. Connectivity Checks 10
2.3. Nominating Candidate Pairs And Concluding ICE 12
2.4. ICE Restart 12
2.5. Lite Implementations 13
3. ICE Usage 13
4. Terminology 13
5. ICE Candidate Gathering and Exchange 17
5.1. Full Implementation 18
5.1.1. Gathering Candidates 18
5.1.1.1. Host Candidates 18
5.1.1.2. Server Reflexive and Relayed Candidates 20
5.1.1.3. Computing Foundations 21
5.1.1.4. Keeping Candidates Alive 21
5.1.2. Prioritizing Candidates 22
5.1.2.1. Recommended Formula 22
5.1.2.2. Guidelines for Choosing Type and Local Preferences 23
5.1.3. Eliminating Redundant Candidates 23
5.2. Lite Implementation Procedures 23
5.3. Exchanging Candidate Information 25
5.4. ICE Mismatch 26
6. ICE Candidate Processing 26
6.1. Procedures for Full Implementation 26
6.1.1. Determining Role 26
6.1.2. Forming the Check Lists 28

6.1.2.1. Check List State 28
6.1.2.2. Forming Candidate Pairs 29
6.1.2.3. Computing Pair Priority and Ordering Pairs . . . 31
6.1.2.4. Pruning the Pairs 31
6.1.2.5. Removing lower-priority Pairs 31
6.1.2.6. Computing Candidate Pair States 32
6.1.3. ICE State 35
6.1.4. Scheduling Checks 35
6.1.4.1. Triggered Check Queue 35
6.1.4.2. Performing Connectivity Checks 36
6.2. Lite Implementation Procedures 37
7. Performing Connectivity Checks 37
7.1. STUN Extensions 38
7.1.1. PRIORITY 38
7.1.2. USE-CANDIDATE 38
7.1.3. ICE-CONTROLLED and ICE-CONTROLLING 38
7.2. STUN Client Procedures 38
7.2.1. Creating Permissions for Relayed Candidates 38
7.2.2. Forming Credentials 39
7.2.3. DiffServ Treatment 39
7.2.4. Sending the Request 39
7.2.5. Processing the Response 39
7.2.5.1. Role Conflict 40
7.2.5.2. Failure 40
7.2.5.2.1. Non-Symmetric Transport Addresses 40
7.2.5.2.2. ICMP Error 41
7.2.5.2.3. Timeout 41
7.2.5.2.4. Unrecoverable STUN Response 41
7.2.5.3. Success 41
7.2.5.3.1. Discovering Peer Reflexive Candidates 41
7.2.5.3.2. Constructing a Valid Pair 42
7.2.5.3.3. Updating Candidate Pair States 43
7.2.5.3.4. Updating the Nominated Flag 43
7.2.5.4. Check List State Updates 44
7.3. STUN Server Procedures 44
7.3.1. Additional Procedures for Full Implementations . . . 45
7.3.1.1. Detecting and Repairing Role Conflicts 45
7.3.1.2. Computing Mapped Address 46
7.3.1.3. Learning Peer Reflexive Candidates 46
7.3.1.4. Triggered Checks 47
7.3.1.5. Updating the Nominated Flag 48
7.3.2. Additional Procedures for Lite Implementations . . . 49
8. Concluding ICE Processing 49
8.1. Procedures for Full Implementations 49
8.1.1. Nominating Pairs 49
8.1.2. Updating States 50
8.2. Procedures for Lite Implementations 51
8.3. Freeing Candidates 52

8.3.1. Full Implementation Procedures 52
8.3.2. Lite Implementation Procedures 52
9. ICE Restarts 52
10. ICE Option 53
11. Keepalives 53
12. Data Handling 54
12.1. Sending Data 54
12.2. Procedures for Lite Implementations 55
12.3. Procedures for All Implementations 55
13. Receiving Data 56
14. Extensibility Considerations 56
15. Setting Ta and RTO 57
15.1. General 57
15.2. Ta 58
15.3. RTO 59
16. Example 60
17. Security Considerations 64
17.1. Attacks on Connectivity Checks 65
17.2. Attacks on Server Reflexive Address Gathering 67
17.3. Attacks on Relayed Candidate Gathering 68
17.4. Insider Attacks 68
17.4.1. STUN Amplification Attack 68
18. STUN Extensions 69
18.1. New Attributes 69
18.2. New Error Response Codes 70
19. Operational Considerations 70
19.1. NAT and Firewall Types 70
19.2. Bandwidth Requirements 71
19.2.1. STUN and TURN Server Capacity Planning 71
19.2.2. Gathering and Connectivity Checks 71
19.2.3. Keepalives 72
19.3. ICE and ICE-lite 72
19.4. Troubleshooting and Performance Management 72
19.5. Endpoint Configuration 73
20. IANA Considerations 73
20.1. STUN Attributes 73
20.2. STUN Error Responses 73
20.3. ICE Options 73
21. IAB Considerations 74
21.1. Problem Definition 74
21.2. Exit Strategy 75
21.3. Brittleness Introduced by ICE 75
21.4. Requirements for a Long-Term Solution 76
21.5. Issues with Existing NAPT Boxes 77
22. Changes from RFC 5245 77
23. Acknowledgements 78
24. References 78
24.1. Normative References 78

24.2. Informative References 79
Appendix A. Lite and Full Implementations 82
Appendix B. Design Motivations 83
B.1. Pacing of STUN Transactions 84
B.2. Candidates with Multiple Bases 85
B.3. Purpose of the Related Address and Related Port Attributes 87
B.4. Importance of the STUN Username 87
B.5. The Candidate Pair Priority Formula 89
B.6. Why Are Keepalives Needed? 89
B.7. Why Prefer Peer Reflexive Candidates? 90
B.8. Why Are Binding Indications Used for Keepalives? 90
B.9. Selecting Candidate Type Preference 90
Appendix C. Connectivity Check Bandwidth 91
Authors' Addresses 92

1. Introduction

Protocols establishing communication sessions between peers typically involve exchanging IP addresses and ports for the data sources and sinks. However this poses challenges when operated through Network Address Translators (NATs) [RFC3235]. These protocols also seek to create a data flow directly between participants, so that there is no application layer intermediary between them. This is done to reduce data latency, decrease packet loss, and reduce the operational costs of deploying the application. However, this is difficult to accomplish through NATs. 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 NATs. 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 data 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 data 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, there are two endpoints (ICE agents) that want to communicate. Note that ICE is not intended for NAT traversal for the signaling protocol, which is assumed to be provided via another mechanism. ICE assumes that the agents are able to establish a signaling connection between each other.

Initially, the agents are ignorant of their own topologies. In particular, the agents may or may not be behind NATs (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 establish a data session.

Figure 1 shows a typical ICE deployment. The agents are labelled L and R. 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. L and R are capable of engaging in an candidate exchange process, whose purpose is to set up a data session between L and R. Typically, this exchange will occur through a signaling server (e.g., SIP proxy).

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.

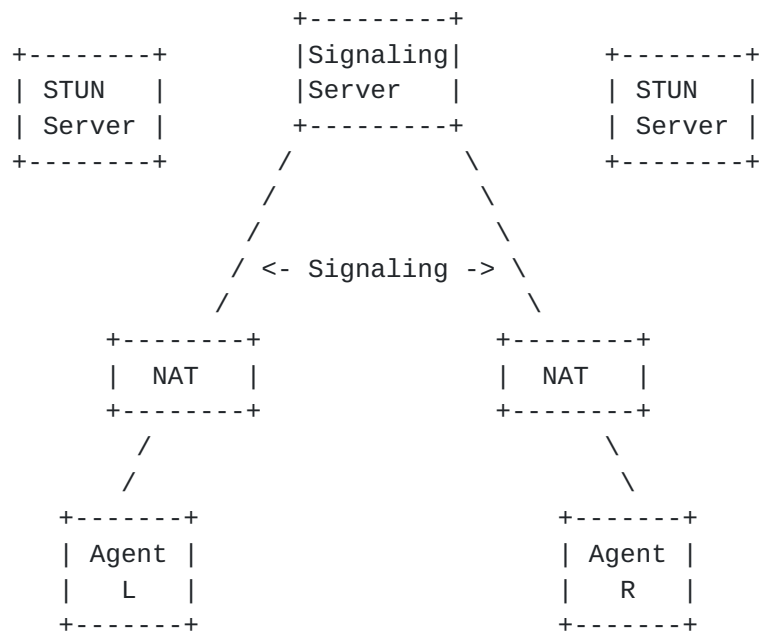


Figure 1: ICE Deployment Scenario

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 Candidates

In order to execute ICE, an ICE agent has to identify all of its address candidates. A candidate has a transport address -- a combination of IP address and port for a particular transport protocol (with only UDP specified here). There are different types of candidates, some derived from physical or logical network interfaces, others discoverable via STUN and TURN. Naturally, one viable candidate has 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.

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.

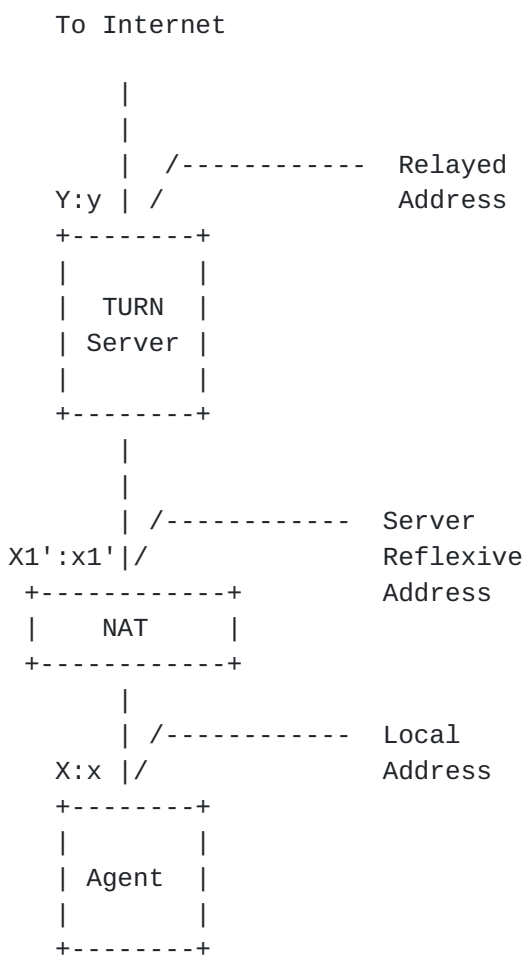


Figure 2: Candidate Relationships

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. The host candidate associated with a given server reflexive candidate is 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 ICE 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 connectivity 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:

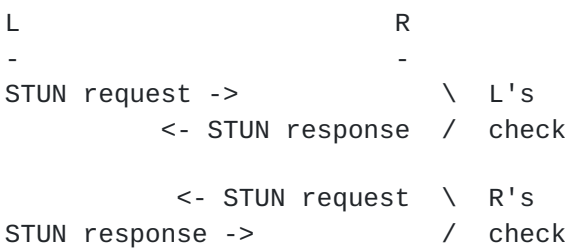


Figure 3: Basic Connectivity Check

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 data (e.g., RTP, RTCP, or other protocols). Consequently, agents demultiplex STUN and data using the contents of the packets, rather than the port on which they are received.

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 that of other candidates the agent already learned, it represents a new candidate (peer reflexive candidate), which then gets tested by ICE just the same as any other candidate.

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 agent works through the check list by sending a STUN request for the next candidate pair on the list periodically. These are called ordinary checks.

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.

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 data relays and through fewer NATs) are preferred over indirect ones (ones with more data relays and more NATs). Within those guidelines, however, agents have a fair amount of discretion about how to tune their algorithms.

A data stream might consist of multiple components (pieces of a data stream that require their own set of candidates, e.g., RTP and RTCP).

2.3. Nominating Candidate Pairs And Concluding ICE

ICE assigns one of the ICE agents in the role of the controlling agent, and the other of the controlled agent. For each component of a data stream, the controlling agent nominates a candidate pair from the valid candidate pairs to be used for data. The exact timing of the nomination is based on local policy.

When nominating, the controlling agent lets the checks continue until at least one valid candidate pair for each component of a data stream is found and then picks a candidate pair from the valid candidate pairs and sends a STUN request on the pair, using an attribute to indicate to the controlled peer that it has nominated the pair. This is shown in Figure 4.

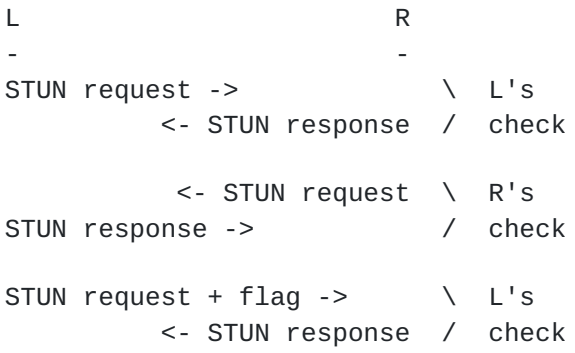


Figure 4: Nomination

Once the controlled agent receives the STUN request with the attribute, it will check (unless the check has already been done) the same pair. If the transactions above succeed, the agents will set the nominated flag for the pairs, and will cancel any future checks for that component of the data stream. Once an agent has set the nominated flag for each component of a data stream, the pairs become the selected pairs. After that, only the selected pairs will be used for sending and receiving data associated with that data stream.

2.4. ICE Restart

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

2.5. Lite Implementations

Certain ICE 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). Lite agents only use host candidates and do not generate connectivity checks or run the state machines, though they need to be able to respond to connectivity checks.

3. ICE Usage

This document specifies generic use of ICE with protocols that provide means to exchange candidate information between the ICE agents. The specific details of (i.e how to encode candidate information and the actual candidate exchange process) for different protocols using ICE (referred to as using protocol) are described in separate usage documents.

One mechanism for agents to exchange the candidate information by using [RFC3264] based Offer/Answer semantics as part of the SIP [RFC3261] protocol [I-D.ietf-mmusic-ice-sip-sdp].

4. 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 Session: An ICE session consists of all ICE-related actions starting with the candidate gathering, followed by the interactions (candidate exchange, connectivity checks, nominations and keep-alives) between the ICE agents until all the candidates are released or ICE-restart is triggered.

ICE Agent, Agent: An ICE agent (sometimes simply referred to as 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 an ICE agent that initiates the ICE candidate exchange process.

Responding Peer, Responding Agent, Responder: A receiving agent is an ICE agent that receives and responds to the candidate exchange process initiated by the initiating agent.

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 ICE 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.

Data, Data Stream, Data Session: When ICE is used to setup data sessions, the data is transported using some protocol. Media is usually transported over RTP, composed of a stream of RTP packets. Data session refers to data packets that are exchanged between the peer on the path created and tested with ICE.

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

Component: A component is a piece of a data stream. A data stream may require multiple components, each of which has to work in order for the data stream as a whole to work. For RTP/RTCP data streams, unless RTP and RTCP are multiplexed in the same port, there are two components per data stream -- one for RTP, and one for RTCP. A component has a candidate pair, which cannot be used by other components

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).

Server Reflexive Candidate: A candidate whose IP address and port are a binding allocated by a NAT for an ICE agent when it sent a packet through the NAT to a server, such as a STUN server.

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

Relayed Candidate: A candidate obtained from a relay server, such as a TURN server.

Base: The transport address that an ICE agent sends from for a particular candidate. For host, server reflexive and peer reflexive candidates the base is the same as the host candidate. For relayed candidates the base is the same as the relayed candidate (i.e., the transport address used by the TURN server to send from).

Foundation: An arbitrary string used in the freezing algorithm to group similar candidates. 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.

Local Candidate: A candidate that an ICE agent has obtained and may send to its peer.

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

Default Destination/Candidate: The default destination for a component of a data stream is the transport address that would be used by an ICE 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 pair of a local candidate and a remote candidate.

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

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

Ordinary Check: A connectivity check generated by an ICE 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 data stream that have been validated by a successful STUN transaction.

Check List Set: The ordered list of all check lists. The order is determined by each ICE usage.

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

Lite Implementation: 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 nominates a candidate pair. In any session, one agent is always controlling. The other is the controlled agent.

Controlled Agent: The ICE agent that waits for the controlling agent to nominate a candidate pair.

Nomination, Regular Nomination: The process of the controlling agent indicating to the controlled agent which candidate pair the ICE agents should use for sending and receiving data.

Nominated, Nominated Flag: Once the nomination of a candidate pair has succeeded, the candidate pair has become nominated, and the value of its nominated flag is set to true.

Selected Pair, Selected Candidate Pair: The candidate pair used for sending and receiving data for a component of a data stream is referred to as the selected pair. Before selected pairs have been produced for a data stream, any valid candidate pair associated with a component of a data stream can be used for sending and receiving data for the component. Once there are nominated pairs for each component of a data stream, the nominated pairs become the selected pairs for the data stream. The candidates associated with the selected pairs are referred to as selected candidates.

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.

5. 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 the encoding mechanism and the semantics of candidate information exchange is out of scope of this specification.

However at a higher level, the diagram below shows how the ICE agents (initiator and responder) exchange their respective candidate(s) information.

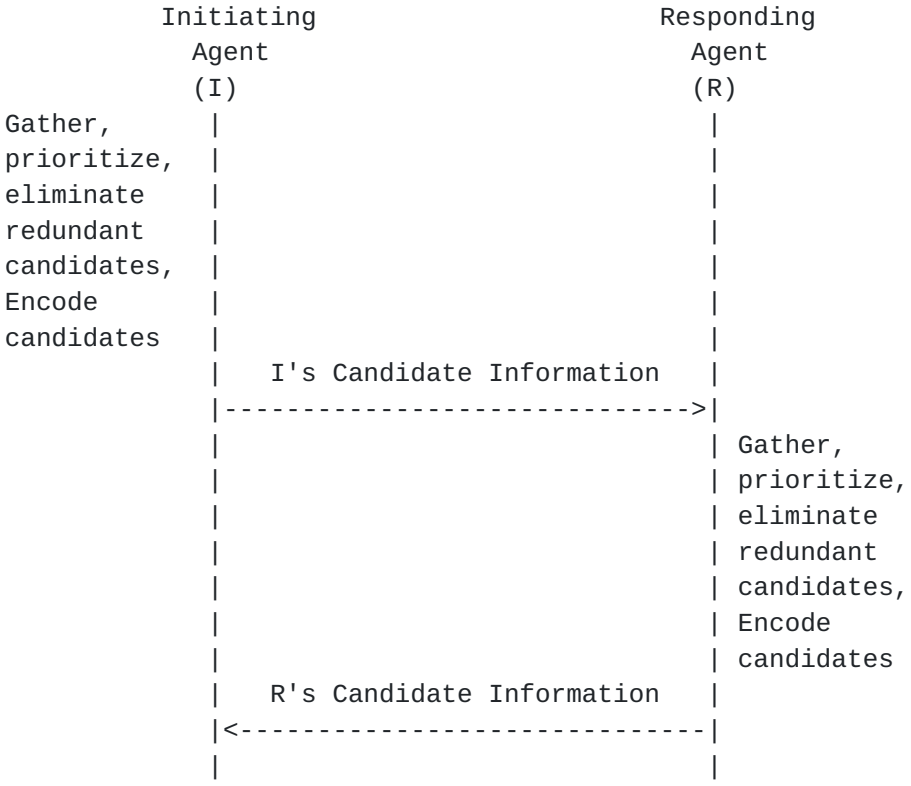


Figure 5: Candidate Gathering and Exchange Sequence

As shown, the agents involved in the candidate exchange perform (1) candidate gathering, (2) candidate prioritization, (3) redundant candidate elimination, (4) (possibly) default candidate selection, and (5) sending of the candidates to the peer. All but the last of these five steps differ for full and lite implementations.

5.1. Full Implementation

5.1.1. Gathering Candidates

An ICE 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 has 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 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 of the application associated with the ICE session. Such gathering MAY begin when an agent starts.

5.1.1.1. Host Candidates

Host candidates are obtained by binding to ports on an IP address attached to an interface (physical or virtual, including VPN interfaces) on the host.

For each component of each data stream the ICE agent wishes to use, the agent SHOULD obtain a candidate 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/RTCP data 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. 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.

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/RTCP 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 address 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 [RFC7721] 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.

- Host candidates corresponding to IPv6 link-local addresses MUST NOT be gathered.

The IPv6 default address selection specification [RFC6724] specifies that temporary addresses [RFC4941] are to be preferred over permanent addresses.

5.1.1.2. Server Reflexive and Relayed Candidates

An ICE agent SHOULD gather server reflexive and relayed candidates. These requirements are at SHOULD strength to allow for provider variation. Use of STUN and TURN servers may be unnecessary in certain networks and use of TURN servers may be expensive, so some deployments may elect not to use them. 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.

The agent pairs each host candidate with the STUN or TURN servers with which it is configured or has discovered by some means. 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.

When multiple STUN or TURN servers are available (or when they are learned through DNS records and multiple results are returned), the agent MAY gather candidates for all of them and SHOULD gather candidates for at least one of them (one STUN server and one TURN server). It does so by pairing host candidates with STUN or TURN servers and, for each pair, the agent sends a Binding or Allocate request to the server from the 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.

The gathering process is controlled using a timer, Ta. Every time Ta expires, 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 time Ta expires. See Section 15 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 also gather 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.

5.1.1.3. Computing Foundations

The ICE agent assigns each candidate a foundation. Two candidates MUST have the same foundation when all of the following are true:

- They have 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).

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.

5.1.1.4. 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 8.3. 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.

Host candidates do not time out, but the candidate addresses may change or disappear for a number of reasons. An ICE agent SHOULD monitor the interfaces it uses, invalidate candidates whose base has gone away, and acquire new candidates as appropriate when new interfaces appear.

5.1.2. Prioritizing Candidates

The prioritization process results in the assignment of a priority to each candidate. Each candidate for a data 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 ICE agent SHOULD compute this priority using the formula in Section 5.1.2.1 and choose its parameters using the guidelines in Section 5.1.2.2. If an agent elects to use a different formula, ICE may take longer to converge since the agents will not be coordinated in their checks.

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

5.1.2.1. Recommended Formula

The recommended formula combines a preference for the candidate type (server reflexive, peer reflexive, relayed, and host), a preference for IP address for which the candidate was obtained, and component ID 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 (lowest preference) to 126 (highest preference) inclusive and 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. Setting the value to 0 means that candidates of this type will only be used as a last resort. Note that candidates gathered based on the procedures of Section 5.1.1 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 (lowest preference) to 65535 (highest preference) inclusive. When there is only a single IP address, this value SHOULD be set to 65535. If there are multiple candidates for a particular component for a particular data stream that have the same type, the local preference MUST be unique for each one. If an ICE agent is dual-stack, the local preference should be set according to the current best practice described in [I-D.ietf-ice-dualstack-fairness].

The component ID MUST be an integer between 1 and 256 inclusive.

5.1.2.2. Guidelines for Choosing Type and Local Preferences

The RECOMMENDED values for type preferences are 126 for host candidates, 110 for peer reflexive candidates, 100 for server reflexive candidates, and 0 for relayed candidates.

If an ICE agent is multihomed and has multiple IP addresses, the recommendations in [I-D.ietf-ice-dualstack-fairness] SHOULD be followed. 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 a preference among different servers but the preference MUST be unique for each one.

When choosing type preferences, agents may take into account factors such as latency, packet loss, cost, network topology, security, privacy, and others.

5.1.3. Eliminating Redundant Candidates

Next, the ICE agents (initiating and responding) eliminate redundant candidates. 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. A candidate is redundant if and only if its transport address and base equal those of another candidate. The agent SHOULD eliminate the redundant candidate with the lower priority.

5.2. Lite Implementation Procedures

Lite implementations only utilize host candidates. A lite implementation MUST, for each component of each data 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 data stream, if an ICE 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/RTCP data 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 data 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 data stream. If a host is IPv4-only, there would only be one candidate for each component of each data 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.

5.3. Exchanging Candidate Information

ICE agents (initiating and responding) need the following information about candidates to be exchanged. Each ICE usage MUST define how the information is exchanged with the using protocol. This section describes the information that needs to be exchanged.

Candidates: One or more candidates. For each candidate:

- Address: The IP address and transport protocol port of the candidate.
- Transport: The transport protocol of the candidate. This MAY be omitted if the using protocol only runs over a single transport protocol.
- Foundation: A sequence of up to 32 characters.
- Component ID: The component ID of the candidate. This MAY be omitted if the using protocol does not use the concept of components.
- Priority: The 32-bit priority of the candidate.
- Type: The type of the candidate.
- Related Address and Port: The related IP address and port of the candidate. 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 or Full: Whether the agent is a lite agent or full agent.

Connectivity check pacing value: The pacing value for connectivity checks that the agent wishes to use. This MAY be omitted if the agent wishes to use a defined default value.

Username Fragment and Password: Values used to perform connectivity checks. The username fragment MUST contain at least 24 bits of randomness, and the password MUST contain at least 128 bits of randomness.

Extensions: New media-stream or session-level attributes (ice-options).

If the using protocol is vulnerable to, and able to detect, ICE mismatch (Section 5.4), a way is needed for the detecting agent to convey this information to its peer. It is a boolean flag.

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.

Once an agent has sent its candidate information, it MUST be prepared to receive both STUN and data packets on each candidate. As discussed in Section 12.1, data packets can be sent to a candidate prior to its appearance as the default destination for data.

5.4. ICE Mismatch

Certain middleboxes, such as ALGs, can alter signaling information in ways that break ICE. This is referred to as ICE mismatch. If the using protocol is vulnerable to ICE mismatch, the responding agent needs to be able to detect it and inform the peer ICE agent about the ICE mismatch.

Each using protocol needs to define whether the using protocol is vulnerable to ICE mismatch, how ICE mismatch is detected, and whether specific actions need to be taken when ICE mismatch is detected.

6. ICE Candidate Processing

Once an ICE agent has gathered its candidates and exchanged candidates with its peer (Section 5), it will determine its own role. In addition, full implementations will form check lists, and begin performing connectivity checks with the peer.

6.1. Procedures for Full Implementation

6.1.1. Determining Role

For each session, each ICE 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 data 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 data 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 8.1 to nominate pairs that will become (if the connectivity checks associated with the nominations succeed) the selected pairs, and then both agents end ICE as described in Section 8.1.2.

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 8.1 to nominate pairs that will become (if the connectivity checks associated with the nominations succeed) the selected pairs, and use the logic in Section 8.1.2 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 8.2. For the lite implementation, the state of ICE processing for each data 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 8 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 data stream is considered to be Running, and the state of ICE overall is Running.

Once the roles are determined for a session, they persist throughout the lifetime of the session. The roles can be re-determined as part of an ICE restart (Section 9), but an ICE agent MUST NOT re-determine the role as part of an ICE restart unless one or more of the following criteria is fulfilled:

Full becomes lite: If the controlling agent is full, and switches to lite, the roles MUST be re-determined if the peer agent is also full.

Role conflict: If the ICE restart causes a role conflict, the roles might be re-determined due to the role conflict procedures in Section 7.3.1.1.

NOTE: There are certain 3PCC scenarios where an ICE restart might cause a role conflict.

NOTE: The agents needs to inform each other whether they are full or lite before the roles are determined. The mechanism for that is signalling protocol specific, and outside the scope of the document.

An agent MUST be prepared that the peer might re-determine the roles as part of any ICE restart, even if the criteria for doing so are not fulfilled. This can happen if the peer is compliant with an older version of this specification.

6.1.2. Forming the Check Lists

There is one check list for each data stream. To form a check list, an ICE agent (initiating and responding) forms candidate pairs, computes pair priorities, orders pairs by priority, prunes pairs, removes lower-priority pairs, and sets check list states. If candidates are added to a check list (e.g, due to detection of peer reflexive candidates), the agent will re-perform these steps for the updated check list.

6.1.2.1. Check List State

Each check list has a state, which captures the state of ICE checks for the data stream associated with the check list. The states are:

Running: The check list is neither Completed nor Failed yet. Check lists are initially set to the Running state.

Completed: The check list contains a nominated pair for each component of the data stream.

Failed: The check list does not have a valid candidate pair for each component of the data stream and all of the candidate pairs in the check list are in either the Failed or Succeeded state. In other words, at least one component of the check list has candidate pairs that are all in the Failed state, which means the component has failed, which means the check list has failed.

6.1.2.2. Forming Candidate Pairs

The ICE agent pairs each local candidate with each remote candidate for the same component of the same data stream with the same IP address family. 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 data stream. If this happens, the number of components for that data 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 data 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 the default candidate pair for that component. This is the pair that would be used to transmit data if both agents had not been ICE aware.

Figure 6 shows the properties of and relationships between transport addresses, candidates, candidate pairs, and check lists.

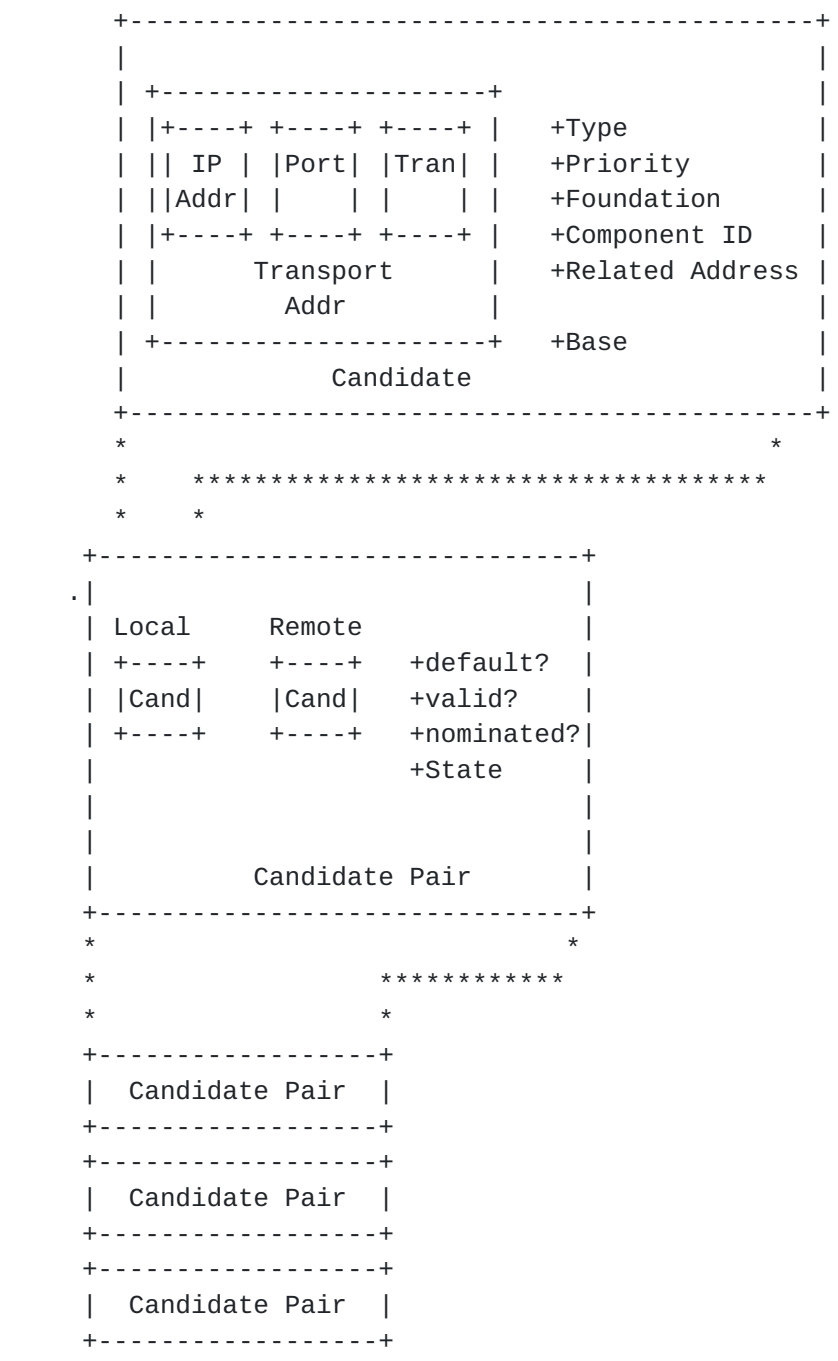


Figure 6: Conceptual Diagram of a Check List

6.1.2.3. Computing Pair Priority and Ordering Pairs

The ICE agent computes a priority for each candidate pair. 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 follows, where G>D?1:0 is an expression whose value is 1 if G is greater than D, and 0 otherwise.

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

The agent sorts each check list in decreasing order of candidate pair priority. If two pairs have identical priority, the ordering amongst them is arbitrary.

6.1.2.4. 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 ICE agent cannot send requests directly from a reflexive candidate (server reflexive or peer reflexive), but only from its base, the agent next goes through the sorted list of candidate pairs. For each pair where the local candidate is reflexive, the candidate MUST be replaced by its base.

The agent prunes each check list. This is done by removing a candidate pair if it is redundant with a higher priority candidate pair in the same check list. Two candidate pairs are redundant if their local candidates have the same base and their remote candidates are identical. The result is a sequence of ordered candidate pairs, called the check list for that data stream.

6.1.2.5. Removing lower-priority Pairs

In order to limit the attacks described in Section 17.4.1, an ICE agent MUST limit the total number of connectivity checks the agent performs across all check lists in the check list set. This is done by limiting the total number of candidate pairs in the check list set. The default limit of candidate pairs for the check list set is 100, but the value MUST be configurable. The limit is enforced by, within in each check list, discarding lower-priority candidate pairs until the total number of candidate pairs in the check list set is smaller than the limit value. The discarding SHOULD be done evenly so that the number of candidate pairs in each check list is reduced the same amount.

It is RECOMMENDED that a lower limit value than the default is picked when possible, and that the value is set to the maximum number of plausible candidate pairs that might be created 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.

6.1.2.6. Computing Candidate Pair States

Each candidate pair in the check list has a foundation (the combination of the foundations of the local and remote candidates in the pair) and one of the following states:

Waiting: A check has not been sent for this pair, but the pair is not Frozen.

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

Succeeded: A check has been sent for this pair, and produced a successful result.

Failed: A check has been sent for this pair, and failed (a response to the check was never received, or a failure response was received).

Frozen: A check for this pair has not been sent, and it can not be sent until the pair is unfrozen and moved into the Waiting state.

Pairs move between states as shown in Figure 7.

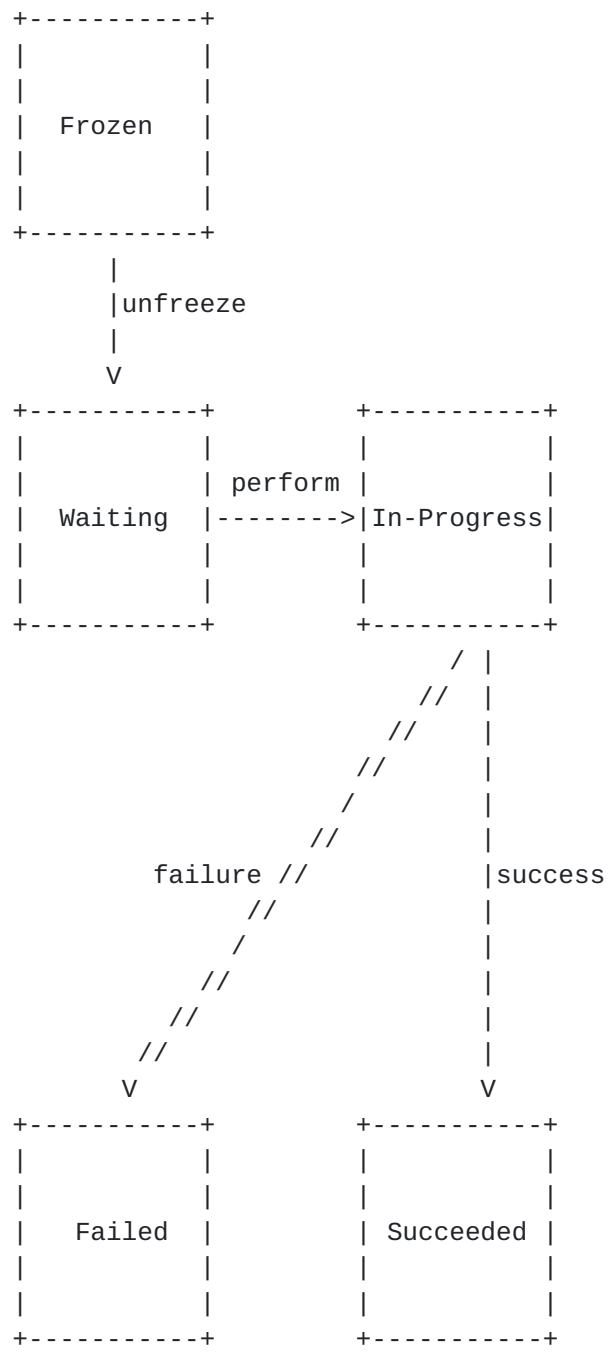


Figure 7: Pair State FSM

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

2. The check lists are placed in an ordered list (the order is determined by each ICE usage), called the check list set.

3. The ICE agent initially places all candidate pairs in the Frozen state.

4. The agent sets all of the check lists in the check list set to the Running state.

5. For each foundation, the agent sets the state of exactly one candidate pair to the Waiting state (unfreezing it). The candidate pair to unfreeze is chosen by finding the first candidate pair (ordered by lowest component ID and then highest priority if component IDs are equal) in the first check list (according to the usage-defined check list set order) that has that foundation.

NOTE: The procedures above are different from RFC5245, where only candidate pairs in the first check list of were initially placed in the Waiting state. Now it applies to candidate pairs in the the first check list which have that foundation, even if the first check list to have that foundation is not the first check list in the check list set.

The table in Figure 8 illustrates an example.

Table legend:

Each row (m1, m2,...) represents a check list associated with a data stream. m1 represents the first check list in the check list set.

Each column (f1, f2,...) represents a foundation. Every candidate pair within a given column share the same foundation.

f-cp represents a candidate pair in the Frozen state.

w-cp represents a candidate pair in the Waiting state.

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

```
      f1    f2    f3    f4    f5
    ----------------------------
m1 | f-cp  f-cp  f-cp
   |
m2 | f-cp  f-cp  f-cp  f-cp
   |
m3 | f-cp                    f-cp
```

2. For each foundation, the candidate pair with the lowest component ID is placed in the Waiting state, unless a candidate pair associated with the same foundation has already been put in the Waiting state in one of the other examined check lists in the check list set.

```
      f1    f2    f3    f4    f5
    ------------------------------
m1 | w-cp  w-cp  w-cp
   |
m2 | f-cp  f-cp  f-cp  w-cp
   |
m3 | f-cp                    w-cp
```

In the first check list (m1) the candidate pair for each foundation is placed in the Waiting state, as no pairs for the same foundations have yet been placed in the Waiting state.

In the second check list (m2) the candidate pair for foundation f4 is placed in the Waiting state. The candidate pair for foundations f1, f2 and f3 are kept in the Frozen state, as candidate pairs for those foundations have already been placed in the Waiting state (within check list m1).

In the third check list (m3) the candidate pair for foundation f5 is placed in the Waiting state. The candidate pair for foundation f1 is kept in the Frozen state, as a candidate pair for that foundation have already been placed in the Waiting state (within check list m1).

Once each check list have been processed, one candidate pair for each foundation in the check list set has been placed in the Waiting state.

Figure 8: Initial Pair State

6.1.3. ICE State

The ICE agent has a state determined by the state of the check lists. The state is Completed if all check lists are Completed, Failed if all check lists are Failed, and Running otherwise.

6.1.4. Scheduling Checks

6.1.4.1. Triggered Check Queue

Once the ICE agent has computed the check lists and created the check list set, as described in Section 6.1.2, the agent will begin performing connectivity checks (ordinary and triggered). For

triggered connectivity checks, the agent maintains a FIFO queue for each check list, referred to as the triggered check queue, which contains candidate pairs for which checks are to be sent at the next available opportunity. The triggered check queue is initially empty.

6.1.4.2. Performing Connectivity Checks

The generation of ordinary and triggered connectivity checks is governed by timer Ta. As soon as the initial states for the candidate pairs in the check list set have been set, a check is performed for a candidate pair within the first check list in the Running state, following the procedures in Section 7. After that, whenever Ta fires the next check list in the Running state in the check list set is picked, and a check is performed for a candidate within that check list. After the last check list in the Running state in the check list set has been processed, the first check list is picked again. Etc.

Whenever Ta fires, the ICE agent will perform a check for a candidate pair within the picked check list by performing the following steps:

1. If the triggered check queue associated with the check list contains one or more candidate pairs, the agent removes the top pair from the queue, performs a connectivity check on that pair, puts the candidate pair state to In-Progress, and aborts the subsequent steps.

2. If there is no candidate pair in the Waiting state, and if there are one or more pairs in the Frozen state, for each pair in the Frozen state the agent checks the foundation associated with the pair. For a given foundation, if there is no pair (in any check list in the check list set) in the Waiting or In-Progress state, the agent puts the candidate pair state to Waiting and continues with the next step.

3. If there are one or more candidate pairs in the Waiting state, the agent picks the highest-priority candidate pair (if there are multiple pairs with the same priority, the pair with the lowest component ID is picked) in the Waiting state, performs a connectivity check on that pair, puts the candidate pair par state to In-Progress, and abort the subsequent steps.

4. If this step is reached, no check could be performed for the picked check list. So, without waiting for timer Ta to expire again, select the next check list in the Running state and return to step #1. If this happens for every single check list in the Running state, meaning there are no remaining candidate pairs to perform connectivity checks for, abort these steps.

Once the agent has picked a candidate pair, for which a connectivity check is to be performed, the agent performs the check by sending a STUN request from the base associated with the local candidate of the pair to the remote candidate of the pair, as described in Section 7.2.4.

Based on local policy, an agent MAY choose to terminate performing the connectivity checks for one or more checks lists in the check list set at any time. However, only the controlling agent is allowed to conclude ICE (Section 8).

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 check lists. On the other hand the responding agent either performs the triggered or ordinary checks as described above.

6.2. Lite Implementation Procedures

Lite implementations skips most of the steps in Section 6 except for verifying the peer's ICE support and determining its role in the ICE processing.

If the lite implementation is the controlling agent (which will only happen if the peer ICE agent is also a lite implementation), it selects 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 data stream, and no further candidate updates are needed to signal this information.

7. Performing Connectivity Checks

This section describes how connectivity checks are performed.

An ICE agent MUST be compliant to [RFC5389]. A full implementation acts both as a STUN client and a STUN server, while a lite implementation only acts as a STUN server (as it does not generate connectivity checks).

7.1. STUN Extensions

ICE extends STUN by defining new attributes: PRIORITY, USE-CANDIDATE, ICE-CONTROLLED, and ICE-CONTROLLING. The new attributes are formally defined in Section 18.1. This section describes the usage of the new attributes.

The new attributes are only applicable to ICE connectivity checks.

7.1.1. PRIORITY

The priority attribute MUST be included in a Binding request and be set to the value computed by the algorithm in Section 5.1.2 for the local candidate, but with the candidate type preference of peer reflexive candidates.

7.1.2. USE-CANDIDATE

The controlling agent MUST include the USE-CANDIDATE attribute in order to nominate a candidate pair Section 8.1.1. The controlled agent MUST NOT include the USE-CANDIDATE attribute in a Binding request.

7.1.3. ICE-CONTROLLED and ICE-CONTROLLING

The controlling agent MUST include the ICE-CONTROLLING attribute in a Binding request. The controlled agent MUST include the ICE-CONTROLLED attribute in a Binding request.

The content of either attribute are used as tie-breaker values when an ICE role conflict occurs Section 7.3.1.1.

7.2. STUN Client Procedures

7.2.1. 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 ICE 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.

7.2.2. Forming Credentials

A connectivity check Binding request 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 ICE 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 ICE agent L is the Initiating agent and ICE 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).

7.2.3. DiffServ Treatment

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

If multiple DSCP markings are used on the data packets, the agent SHOULD choose one of them for use with the connectivity check.

7.2.4. Sending the Request

A connectivity check is generated by sending a Binding request from the base associated with a local candidate to a remote candidate. [RFC5389] describes how Binding requests are constructed and generated.

Support for backwards compatibility with RFC 3489 MUST NOT be assumed when performing connectivity checks. The FINGERPRINT mechanism MUST be used for connectivity checks.

7.2.5. Processing the Response

This section defines additional procedures for processing Binding responses specific to ICE connectivity checks.

When a Binding response is received, it is correlated to the corresponding Binding request using the transaction ID [RFC5389],

which then associates the response with the candidate pair for which the Binding request was sent. After that, the response is processed according to the procedures for a role conflict, a failure, or a success, according to the procedures below.

7.2.5.1. Role Conflict

If the Binding request generates a 487 (Role Conflict) error response, and if the ICE agent included an ICE-CONTROLLED attribute in the request, the agent MUST switch to the controlling role. If the agent included an ICE-CONTROLLING attribute in the request, the agent MUST switch to the controlled role.

Once the agent has switched its role, the agent MUST add the candidate pair whose check generated the 487 error response to the triggered check queue associated with the check list to which the pair belongs, and set the candidate pair state to Waiting. When the triggered connectivity check is later performed, the ICE-CONTROLLING/ ICE-CONTROLLED attribute of the Binding request will indicate the agent's new role. The agent MAY change the tie-breaker value.

NOTE: A role switch requires an agent to recompute pair priorities (Section 6.1.2.3), since the priority values depend on the role.

NOTE: A role switch will also impact whether the agent is responsible for nominating candidate pairs, and whether the agent is responsible for initiating the exchange of the updated candidate information with the peer once ICE is concluded.

7.2.5.2. Failure

This section describes cases when the candidate pair state is set to Failed.

NOTE: When the ICE agent sets the candidate pair state to Failed as a result of a connectivity check error, the agent does not change the states of other candidate pairs with the same foundation.

7.2.5.2.1. Non-Symmetric Transport Addresses

The ICE agent MUST check that the source and destination transport addresses in the Binding request and response are symmetric. I.e., the source IP address and port of the response MUST be 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 MUST be equal to the source IP address and port from which the Binding request was sent. If the addresses are not symmetric, the agent MUST set the candidate pair state to Failed.

7.2.5.2.2. ICMP Error

An ICE agent MAY support processing of ICMP errors for connectivity checks. If the agent supports processing of ICMP errors, and if a Binging request generates an ICMP error, the agent SHOULD set the state of the candidate pair to Failed.

7.2.5.2.3. Timeout

If the Binding request times out, the ICE agent SHOULD set the candidate pair state to Failed.

7.2.5.2.4. Unrecoverable STUN Response

If the Binding request generates a STUN error response that is unrecoverable [RFC5389] the ICE agent SHOULD set the candidate pair state to Failed.

7.2.5.3. Success

A connectivity check is considered a success if each of the following criteria is true:

- The Binding request generated a success response; and
- The source and destination transport addresses in the Binding request and response are symmetric.

If a check is considered a success, the ICE agent performs (in order) the actions described in the following sections.

7.2.5.3.1. Discovering Peer Reflexive Candidates

The ICE agent MUST check 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, a peer reflexive candidate has a type, base, priority, and foundation. They are computed as follows:

- The type is peer reflexive.
- The base is local candidate of the candidate pair from which the Binding request was sent.
- The priority is the value of the PRIORITY attribute in the Binding request.

- The foundation is described in Section 5.1.1.3.

The peer reflexive candidate is then added to the list of local candidates for the data stream. The username fragment and password are the same as for all other local candidates for that data stream.

The ICE agent does not need to pair the peer reflexive candidate with remote candidates, as a valid candidate pair will be created due to the procedures in Section 7.2.5.3.2. If an agent wishes to pair the peer reflexive candidate with remote candidates other than the one in the valid pair that will be generated, the agent MAY provide updated candidate information to the peer that includes the peer reflexive candidate. This will cause the peer reflexive candidate to be paired with all other remote candidates.

7.2.5.3.2. Constructing a Valid Pair

The ICE 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.

The valid pair may equal the pair that generated the connectivity check, or it may equal a different pair in a check list (sometimes in a different check list than the one to which the pair that generated the connectivity checks), or it may be a pair not currently in any check list.

The agent maintains a separate list, referred to as the valid list. There is a valid list for each check list in the check list set. The valid list will contain valid pairs. Initially each valid list is empty.

Each valid pair within the valid list has a flag, called the nominated flag. When a valid pair is added to a valid list, the flag value is set to 'false'.

The valid pair will be added to a valid list as follows:

1. If the valid pair equals the pair that generated the check, the pair is added to the valid list associated with the check list to which the pair belongs; or

2. If the valid pair equals another pair in a check list, that pair is added to the valid list associated with the check list of that pair. The pair that generated the check is not added to a valid list; or

3. If the valid pair is not in any check list, the agent computes the priority for the pair based on the priority of each candidate, using the algorithm in Section 6.1.2. The priority of the local candidate depends on its type. Unless the type is peer reflexive, the priority is equal to the priority signaled for that candidate in the candidate exchange. If the type 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.

NOTE: It will be very common that the valid pair will not be in any check list. Recall that the check list has pairs whose local candidates are never reflexive; those pairs had their local candidates converted to the base of the reflexive candidates, and then pruned if they were redundant. When the response to the Binding request 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.

7.2.5.3.3. Updating Candidate Pair States

The ICE agent sets the states of both the candidate pair that generated the check and the constructed valid pair (which may be different) to Succeeded.

The agent MUST set the states for all other Frozen candidate pairs in all check lists with the same foundation to Waiting.

NOTE: Within a given check list, candidate pairs with the same foundations will typically have different component ID values.

7.2.5.3.4. Updating the Nominated Flag

If the controlling agent sends a Binding request with the USE-CANDIDATE attribute set, and if the ICE agent receives a successful response to the request, the agent sets the nominated flag of the pair to true. If the request fails Section 7.2.5.2, the agent MUST remove the candidate pair from the valid list, set the candidate pair state to Failed and set the check list state to Failed.

If the controlled agent receives a successful response to a Binding request sent by the agent, and that Binding request was triggered by a received Binding request with the USE-CANDIDATE attribute set

Section 7.3.1.4, the agent sets the nominated flag of the pair to true. If the triggered request fails, the agent MUST remove the candidate pair from the valid list, set the candidate pair state to Failed and set the check list state to Failed.

Once the nominated flag is set for a component of a data stream, it concludes the ICE processing for that component. See Section 8.

7.2.5.4. Check List State Updates

Regardless of whether a connectivity check was successful or failed, the completion of the check may require updating of check list states. For each check list in the check list set, if all of the candidate pairs are in either Failed or Succeeded state, and if there is not a valid pair in the valid list for each component of the data stream associated with the check list, the state of the check list is set to Failed. If there is a valid pair for each component in the valid list, the state of the check list is set to Succeeded.

7.3. STUN Server Procedures

An ICE agent (lite or full) MUST be prepared to receive Binding requests on the base of each candidate it included in its most recent candidate exchange.

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 7.3.1.2). 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 7.3.1.3, Section 7.3.1.4, and Section 7.3.1.5 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 data packets, it SHOULD apply the same markings to Binding responses. The same would apply to any layer 2 markings the endpoint might be applying to data packets.

7.3.1. Additional Procedures for Full Implementations

This subsection defines the additional server procedures applicable to full implementations, when the full implementation accepts the Binding request.

7.3.1.1. Detecting and Repairing Role Conflicts

In certain usages of ICE (such as third party call control), both ICE agents may end up choosing the same role, resulting in a role conflict. The section describes a mechanism for detecting and repairing role conflicts. 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 value 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 value 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 value 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 value 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 6.1.2.3), since those priorities are a function of role. 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 7.3.1 are followed if the agent generated a successful response to the Binding request, even if the agent changed roles.

7.3.1.2. Computing Mapped Address

For requests 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.

7.3.1.3. 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 type is peer reflexive.

- The priority is the value of the PRIORITY attribute in the Binding request.

- The foundation is an arbitrary value, different from the foundations of 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 is the component ID of the local candidate to which the request was sent.

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

7.3.1.4. Triggered Checks

Next, the agent constructs a pair whose local candidate has the transport address (as seen by the agent) 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 7.2.4. 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 picked.

7.3.1.5. Updating the Nominated Flag

If the controlled agent receives a Binding request with the USE-CANDIDATE attribute set, and if the ICE agent accepts the request, the following action is based on the state of the pair computed in Section 7.3.1.4:

- If the state of this pair is Succeeded, it means that the check previously sent by this pair produced a successful response, and generated a valid pair (Section 7.2.5.3.2). The agent sets the nominated flag value of the pair to true.
- If the received Binding request triggered a new check to be enqued in the triggered check queue (Section 7.3.1.4), once the check is sent and if it generates a successful response, and generates a valid pair, the agent sets the nominated flag of the pair to true. If the request fails Section 7.2.5.2, the agent MUST remove the candidate pair from the valid list, set the candidate pair state to Failed and set the check list state to Failed.

If the controlled agent does not accept the request from the controlling agent, the controlled agent MUST reject the nomination request with an appropriate error code response (e.g., 400) [RFC5389].

Once the nominated flag is set for a component of a data stream, it concludes the ICE processing for that component. See Section 8.

7.3.2. Additional Procedures for Lite Implementations

If the controlled agent receives a Binding request with the USE-CANDIDATE attribute set, and if the ICE agent accepts the request, the agent constructs a candidate pair whose local candidate has 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 the valid list of the associated check list. The agent sets the nominated flag for that pair to true.

Once the nominated flag is set for a component of a data stream, it concludes the ICE processing for that component. See Section 8.

8. Concluding ICE Processing

This section describes how an ICE agent completes ICE.

8.1. Procedures for Full Implementations

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

8.1.1. Nominating Pairs

Prior to nominating, the controlling agent let connectivity checks continue until some stopping criterion is met. After that, based on an evaluation criterion, the controlling agent picks a pair among the valid pairs in the valid list for nomination.

Once the controlling agent has picked a valid pair for nomination, it repeats the connectivity 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 Section 7.3.1.5.

Eventually, if the nominations succeed, both the controlling and controlled agents will have a single nominated pair in the valid list for each component of the data stream. Once an ICE agent sets the state of the check list to Completed (when there is a nominated pair for each component of the data stream), that pair becomes the selected pair for that agent, and is used for sending and receiving data for that component of the data stream.

If an agent is not able to produce selected pairs for a data stream, the agent MUST take proper actions for informing the other agent, and e.g., removing the stream. The exact actions are outside the scope of this specification.

The criterion details for stopping the connectivity checks and for selecting a pair for nomination, are outside the scope of this specification. They are a matter of local optimization. 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 set.

If more than one candidate pair is nominated by the controlling agent, and if the controlled agent accepts multiple nominations requests, the agents MUST produce the selected pairs using the pairs with the highest priority.

NOTE: A controlling agent that does not support this specification (i.e. it is implemented according to RFC 5245) might nominate more than one candidate pair. This was referred to as aggressive nomination in RFC 5245. The usage of the 'ice2' ice option Section 10 by endpoints supporting this specification should prevent such controlling agents from using aggressive nomination.

8.1.2. 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 data 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 data stream and the state of the check list is Running:

  * The ICE 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 data 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 data stream and the state of the check list is Running:

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

    * The agent MUST continue to respond to any checks it may still receive for that data stream, and MUST perform triggered checks if required by the processing of Section 7.3.

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

    * The agent MAY begin transmitting data for this data stream as described in Section 12.1.

- 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 data stream. The correct behavior depends on the state of the check lists for other data 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 data streams is Completed, the controlling agent SHOULD remove the failed data stream from the session while sending updated candidate list to its peer.

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

8.2. Procedures for Lite Implementations

When ICE concludes, a lite ICE agent can free host candidates that were not used by ICE, as described in Section 8.3.

If the peer is a full agent, once the lite agent accepts a nomination request for a candidate pair, the lite agent considers the pair nominated. Once there are nominated pairs for each component of a data stream, the pairs become the selected pairs for the components of the data stream. Once the lite agent has produced selected pairs for all components of all data streams, the ICE session state is set to Completed.

If the peer is a lite agent, the agent pairs local candidates with remote candidates that are for the same data stream and have the same

component, transport protocol, and IP address family. For each component of each data stream, if there is only one candidate pair, that pair is added to the valid list. If there is more than one pair, it is RECOMMENDED that an agent follow the procedures of RFC 6724 [RFC6724] to select a pair and add it to the valid list.

If all of the components for all data streams had one pair, the state of ICE processing is Completed. Otherwise, the controlling agent MUST send an updated candidate list to reconcile different agents selecting different candidate pairs. ICE processing is complete after and only after the updated candidate exchange is complete.

8.3. Freeing Candidates

8.3.1. Full Implementation Procedures

The procedures in Section 8 require that an ICE agent continue to listen for STUN requests and continue to generate triggered checks for a data 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 did not become a selected candidate (is not associated with a selected pair), and then free the candidate.

Once a check list has reached the Completed state, the agent SHOULD wait an additional three seconds, and then it can cease responding to checks or generating triggered checks on all local candidates other than the ones that became selected candidates. Once all ICE sessions have ceased using a given local candidate (a candidate may be used by multiple ICE sessions, e.g. in forking scenarios), the agent can free that candidate. The three-second delay handles cases when aggressive nomination is used, and the selected pairs can quickly change after ICE has completed.

Freeing of server reflexive candidates is never explicit; it happens by lack of a keepalive.

8.3.2. Lite Implementation Procedures

A lite implementation can free candidates that did not become selected candidates as soon as ICE processing has reached the Completed state for all ICE sessions using those candidates.

9. ICE Restarts

An ICE agent MAY restart ICE for existing data streams. An ICE restart causes all previous state of the data streams, excluding the roles of the agents to be flushed. The only difference between an

ICE restart and a brand new data session is that during the restart, data can continue to be sent using existing data sessions, and that a new data session always requires the roles to be determined.

The following actions can be accomplished only using an ICE restart (the agent MUST use ICE restarts to do so):

- Change the destinations of data streams.
- Change from a lite implementation to a full implementation.
- Change from a full implementation to a lite implementation.

To restart ICE, an agent MUST change both the password and the username fragment for the data stream(s) being restarted. The new candidate set MAY include some, none, or all of the previous candidates.

As described in Section 6.1.1, agents MUST NOT re-determine the roles as part as an ICE restart, unless certain criteria that require the roles to be re-determined are fulfilled.

10. ICE Option

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

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

11. Keepalives

All endpoints MUST send keepalives for each data session. These keepalives serve the purpose of keeping NAT bindings alive for the data session. The keepalives 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 ICE agent is a full ICE implementation and is communicating with a peer that supports ICE (lite or full).

For each candidate pair that an agent is using to send data, if no packet has been sent on that pair in the last Tr seconds, an agent MUST send a keepalive on that pair. Agents SHOULD use a Tr value of 15 seconds. Agents MAY use a bigger value, but MUST NOT use a value smaller than 15 seconds.

Once selected pairs have been produced for a data stream, keepalives are only sent on those pairs.

An agent MUST stop sending keepalives on a data stream if the data stream is removed. If the ICE session is terminated, an agent MUST stop sending keepalives on all data streams.

An agent MAY use another value for Tr, e.g. based on configuration or network/NAT characteristics. For example, 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.

When 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 data. 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.

Agents MUST by default use STUN keepalives. Individual ICE usages and ICE extensions MAY specify usage/extension-specific keepalives.

12. Data Handling

12.1. Sending Data

An ICE agent MAY send data on any valid candidate pair before selected pairs have been produced for the data stream.

Once selected pairs have been produced for a data stream, an agent MUST send data on those pairs.

An agent sends data from the base of the local candidate to the remote candidate. In the case of a local relayed candidate, data is

forwarded through the base (located in the TURN server), using the procedures defined in [RFC5766].

If the local candidate is a relayed candidate, it is RECOMMENDED that an agent creates 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 data stream is:

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

Unless an agent is able to produce a selected pair for each component associated with a data stream, the agent MUST NOT continue sending data for any component associated with that data stream.

12.2. Procedures for Lite Implementations

A lite implementation MUST NOT send data until it has a valid list that contains a candidate pair for each component of that data stream. Once that happens, the ICE agent MAY begin sending data packets. To do that, it sends data 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 base associated with the candidate pair used for sending data. In case of a relayed candidate, data is sent from the agent and forwarded through the base (located in the TURN server), using the procedures defined in [RFC5766].

12.3. 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, ICE agents are encouraged to re-adjust jitter buffers when there are changes in source or destination address of data 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 data from one candidate pair to another.

13. Receiving Data

Even though ICE agents are only allowed to send data using valid candidate pairs (and, once selected pairs have been produced, only on the selected pairs) ICE implementations SHOULD by default be prepared to receive data on any of the candidates provided in the most recent candidate exchange with the peer. ICE usages MAY define rules that differs from this, e.g., by defining that data must not be sent until selected pairs have been produced for a data stream.

It is RECOMMENDED that, when an agent receives an RTP packet with a new source or destination IP address for a particular RTP/RTCP data 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 data streams switch between candidates. An agent will be able to determine that a data stream is from the same peer as a consequence of the STUN exchange that proceeds media data transmission. Thus, if there is a change in source transport address, but the media data packets come from the same peer agent, this MUST NOT be treated as an SSRC collision.

14. Extensibility Considerations

This specification makes very specific choices about how both ICE agents in a session coordinate to arrive at the set of candidate pairs that are selected for data. 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 8 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 data streams beyond RTP, and for transport protocols beyond UDP. Extensions to ICE for non-RTP data 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.

15. Setting Ta and RTO

15.1. General

During the ICE gathering phase (Section 5.1.1) and while ICE is performing connectivity checks (Section 7), an ICE agent triggers STUN and TURN transactions. These transactions are paced at a rate indicated by Ta, and the retransmission interval for each transaction is calculated based on the the retransmission timer for the STUN transactions (RTO) [RFC5389].

This section describes how the Ta and RTO values are computed during the ICE gathering phase and while ICE is performing connectivity checks.

NOTE: Previously, in RFC 5245, different formulas were defined for computing Ta and RTO, depending on whether ICE was used for a real-time data stream (e.g., RTP) or not.

The formulas below 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.

15.2. Ta

ICE agents SHOULD use the default Ta value, 50 ms, but MAY use another value based on the characteristics of the associated data.

If an agent wants to use another Ta value than the default value, the agent MUST indicate the proposed value to its peer during the establishment of the ICE session. Both agents MUST use the higher value of the proposed values. If an agent does not propose a value, the default value is used for that agent when comparing which value is higher.

Regardless of the Ta value chosen for each agent, the combination of all transactions from all agents (if a given implementation runs several concurrent agents) MUST NOT be sent more often than once every 5ms (as though there were one global Ta value for pacing all agents).

This mechanism of a global minimum pacing interval of 5ms is not generally applicable to transport protocols, but is applicable to ICE based on the following reasoning.

- Start with the following rules which would be generally applicable to transport protocols:

  1. Let MaxBytes be the maximum number of bytes allowed to be outstanding in the network at start-up, which SHOULD be 14600 bytes per RFC 6928.

  2. Let HTO be the transaction timeout, which SHOULD be 2*RTT if RTT is known and 500ms otherwise. This is based on the RTO for STUN messages from RFC 5389 and the the TCP initial RTO, which is 1 sec in RFC 6298.

  3. Let MinPacing be the minimum pacing interval between transactions, which SHOULD be 5ms.

- Observe that agents typically do not know the RTT for ICE transactions (connectivity checks in particular), meaning that HTO will almost always be 500ms.

- Observe that a MinPacing of 5ms and HTO of 500ms gives at most 100 packets/HTO, which for a typical ICE check of less than 120 bytes means a maximum of 12000 outstanding bytes in the network, which is less than the maximum expressed by rule 1.

- Thus, for ICE, the rule set reduces down to just the MinPacing rule, which is equivalent to having a global Ta value.

NOTE: Appendix C shows examples of required bandwidth, using different Ta values.

15.3. RTO

During the ICE gathering phase, ICE agents SHOULD calculate the RTO value using the following formula:

```
RTO = MAX (500ms, Ta * (Num-Of-Pairs))
```

Num-Of-Pairs: the number of pairs of candidates with STUN or TURN servers.

For connectivity checks, agents SHOULD calculate the RTO value using the following formula:

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

Num-Waiting: the number of checks in the check list in the Waiting state.

Num-In-Progress: 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.

Agents MAY calculate the RTO value using other mechanisms than those described above. Agents MUST NOT use a RTO value smaller than 500 ms.

16. Example

The example is based on the simplified topology of Figure 9.

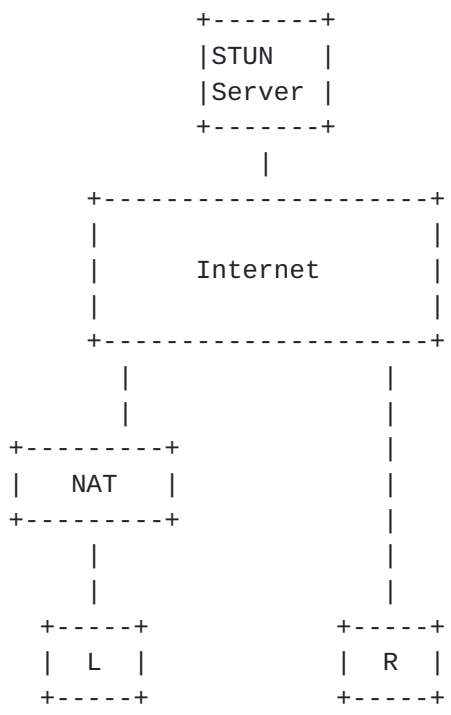


Figure 9: Example Topology

Two ICE agents, L and R, are using ICE. Both are full ICE implementations. 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.

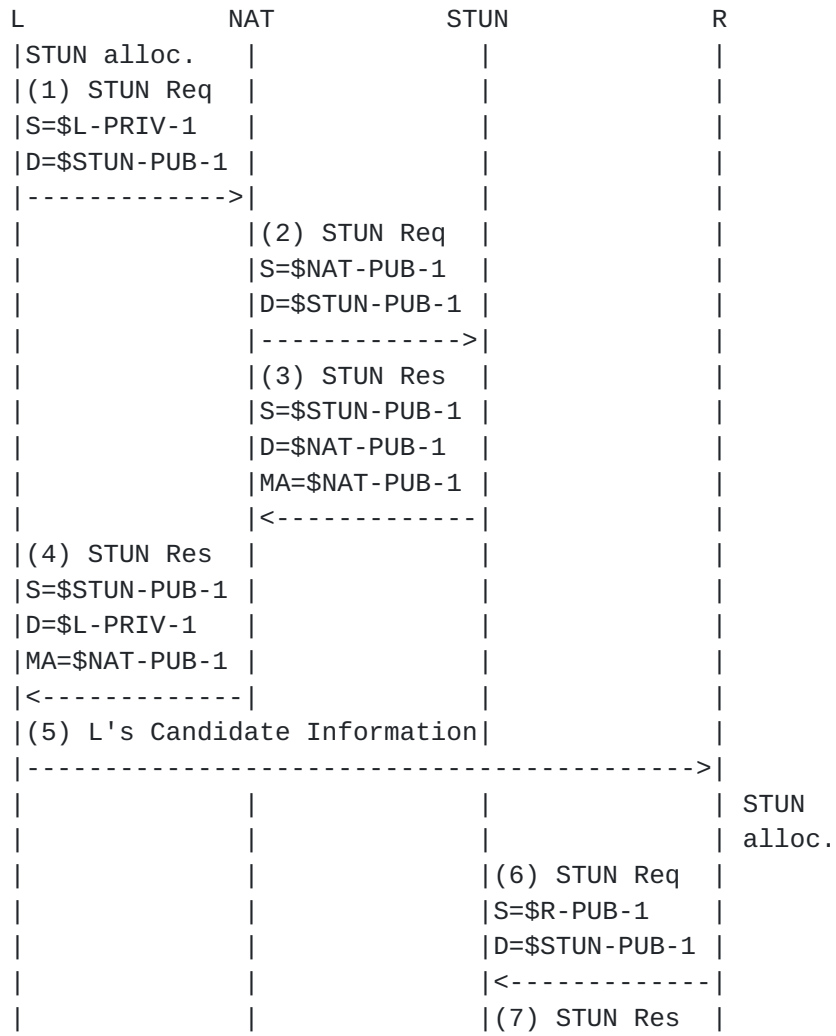
To facilitate understanding, transport addresses are listed using variables that have mnemonic names. The format of the name is entity-type-seqno, where entity refers to the entity whose IP address the transport address is on, and is one of "L", "R", "STUN", or "NAT". The type is either "PUB" for transport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

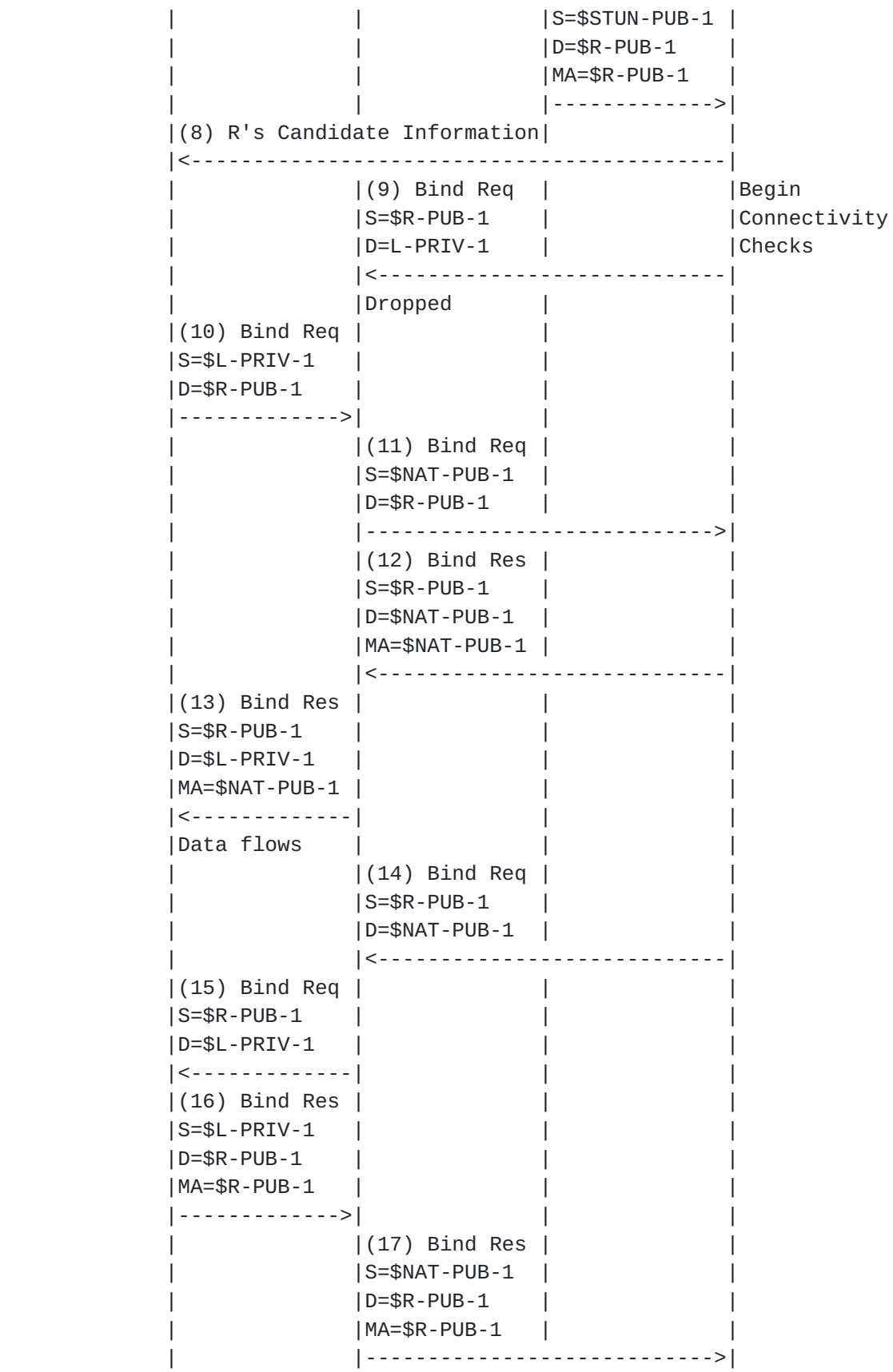
transport address of the same type on a particular entity. Each variable has an IP address and port, denoted by varname.IP and varname.PORT, respectively, where varname is the name of the variable.

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 focus on a single data stream between two full implementations.




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
Begin
Connectivity
Checks
(10) Bind Req
S=$L-PRIV-1
D=$R-PUB-1
(11) Bind Req
S=$NAT-PUB-1
D=$R-PUB-1
(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
Data flows
(14) Bind Req
S=$R-PUB-1
D=$NAT-PUB-1
(15) Bind Req
S=$R-PUB-1
D=$L-PRIV-1
(16) Bind Res
S=$L-PRIV-1
D=$R-PUB-1
MA=$R-PUB-1
(17) Bind Res
S=$NAT-PUB-1
D=$R-PUB-1
MA=$R-PUB-1

```
           |              |              |              |Data flows
```

Figure 10: 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.

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). 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. Agent L can now send data 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 data 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 data.

17. Security Considerations

The process of probing for candidates reveals the source addresses of the client and its peer to any on-network listening attacker, and the process of exchanging candidates reveals the addresses to any attacker that is able to see the negotiation. Some addresses, such as the server reflexive addresses gathered through the local interface of VPN users, may be sensitive information. If these potential attacks can not be mitigated, the implementation may want to institute controls for which addresses are revealed to the negotiation and/or probing process. Such controls need to be specified as part of the ICE usage.

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.

17.1. Attacks on Connectivity Checks

An attacker might attempt to disrupt the STUN connectivity checks. Ultimately, all of these attacks fool an ICE 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 data.

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 data 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 data path is secured (e.g., using SRTP [RFC3711]), the attacker will not be able to process the data packets, but will only be able to discard them, effectively disabling the data stream. 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 data stream, it's much easier to just disrupt it with the same mechanism, rather than attack ICE.

17.2. 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 establishment of the ICE session. For this candidate to actually be used for data, 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 ICE agent in the session, and is prevented by SRTP if it identifies the attacker itself.

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 data. 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.

17.3. 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.

17.4. 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.

17.4.1. 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 data 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. ICE 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.

18. STUN Extensions

18.1. New Attributes

This specification defines four new STUN 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 data. 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. The number is

used for solving role conflicts, when it is referred to as the tie-breaker value. An ICE agent MUST use the same number for all Binding requests, for all streams, within an ICE session. The agent MAY change the number when an ICE restart occurs.

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. The number is used for solving role conflicts, when it is referred to as the tie-breaker value. An agent MUST use the same number for all Binding requests, for all streams, within an ICE session. The agent MAY change the number when an ICE restart occurs.

18.2. 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 an ICE role that conflicted with the server. The server compared the tie-breaker values of the client and the server and determined that the client needs to switch roles.

19. Operational Considerations

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

19.1. 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.

19.2. Bandwidth Requirements

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

19.2.1. 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 data 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 data. The amount of calls requiring TURN for data 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.

19.2.2. 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 data traffic itself. This was done to ensure that, if a network is designed to support communication traffic of a certain type (voice, video, or just text), it will have sufficient capacity to support the ICE checks for that data. 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.

19.2.3. Keepalives

STUN keepalives (in the form of STUN Binding Indications) are sent in the middle of a data session. However, they are sent only in the absence of actual data 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.

19.3. 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.

19.4. 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. Signaling servers, 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 signaling is performed exactly for the purposes of educating network equipment (such as a diagnostic tool attached to a signaling) about the results of ICE processing.

As a consequence, through the logs generated by a signaling server, a network operator can observe what types of candidates are being used

for each call, and what address were selected by ICE. This is the primary information that helps evaluate how ICE is performing.

19.5. 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.

20. 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.

20.1. STUN Attributes

IANA has registered four STUN attributes:

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

20.2. 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.

20.3. 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

21. IAB Considerations

The IAB has studied the problem of "Unilateral Self-Address Fixing", which is the general process by which an ICE 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.

21.1. 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.

21.2. 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 picks 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.

21.3. 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 ICE 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.

21.4. 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.

21.5. 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.

22. Changes from RFC 5245

The purpose of this updated ICE specification is to:

- Clarify procedures in RFC 5245.
- Make technical changes, due to discovered flows in RFC 5245 and based on feedback from the community that has implemented and deployed ICE applications based on RFC 5245.
- Make the procedures signaling protocol independent, by removing the SIP and SDP procedures. Procedures specific to a signaling protocol will be defined in separate usage documents. [I-D.ietf-mmusic-ice-sip-sdp] defines the ICE usage with SIP and SDP.

The following technical changes have been done:

- Aggressive nomination removed.
- The procedures for calculating candidate pair states and scheduling connectivity checks modified.
- Procedures for calculation of Ta and RTO modified.

- Active check list and frozen check list definitions removed.
- 'ice2' ice option added.
- IPv6 considerations modified.
- Usage with no-op for keepalives, and keepalives with non-ICE peers, removed.

23. 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, Suhas Nandakumar, Taylor Brandstetter, Peter Saint-Andre, Harald Alvestrand and Roman Shpount.

24. References

24.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <https://www.rfc-editor.org/info/rfc4941>.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6724>.

24.2. Informative References

[RFC3605] Huitema, C., "Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)", RFC 3605, DOI 10.17487/RFC3605, October 2003, <https://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, <https://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, <https://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, <https://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, <https://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, <https://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, <https://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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc3711>.

[RFC3879] Huitema, C. and B. Carpenter, "Deprecating Site Local Addresses", RFC 3879, DOI 10.17487/RFC3879, September 2004, <https://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, <https://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, <https://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, <https://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, <https://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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc4787>.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, DOI 10.17487/RFC5761, April 2010, <https://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, <https://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, <https://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, <https://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, <https://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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7050>.

[I-D.ietf-mmusic-ice-sip-sdp] Petit-Huguenin, M., Keranen, A., and S. Nandakumar, "Session Description Protocol (SDP) Offer/Answer procedures for Interactive Connectivity Establishment (ICE)", draft-ietf-mmusic-ice-sip-sdp-14 (work in progress), October 2017.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <https://www.rfc-editor.org/info/rfc7721>.

[I-D.ietf-ice-dualstack-fairness] Martinsen, P., Reddy, T., and P. Patil, "ICE Multihomed and IPv4/IPv6 Dual Stack Guidelines", draft-ietf-ice-dualstack-fairness-07 (work in progress), November 2016.

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 ICE 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 17 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

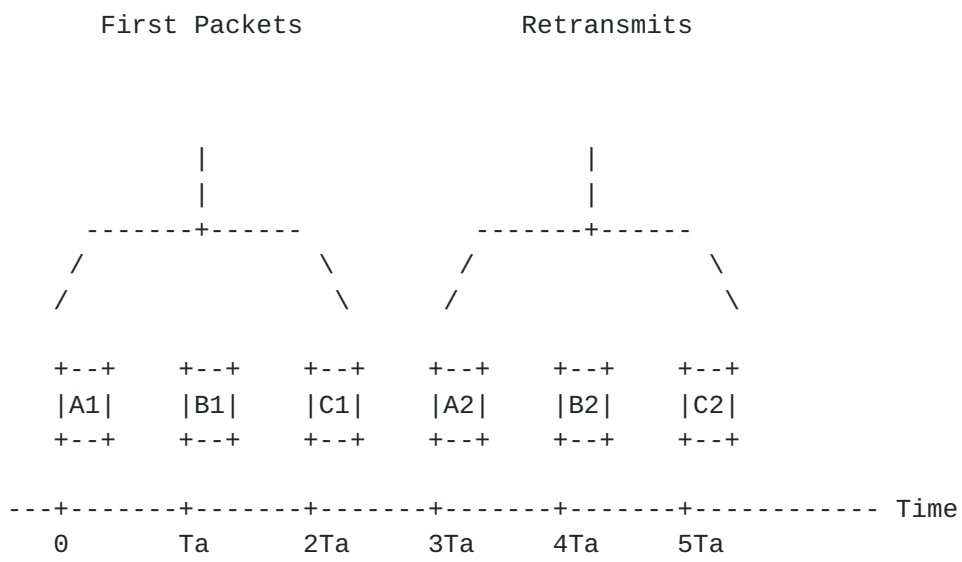
ICE contains a number of normative behaviors that may themselves be simple, but derive from complicated or non-obvious thinking or use cases that merit further discussion. Since these design motivations are not necessary to understand for purposes of implementation, they are discussed here in an appendix 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 5 ms is well supported. This is why Ta has a lower bound of 5 ms. Furthermore, transmission of these packets on the network makes use of bandwidth and needs to be rate limited by the ICE 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 data 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 data 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 5.1.3 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 ICE agent have two candidates that have the same IP address and port, but different bases? Consider the topology of Figure 11:

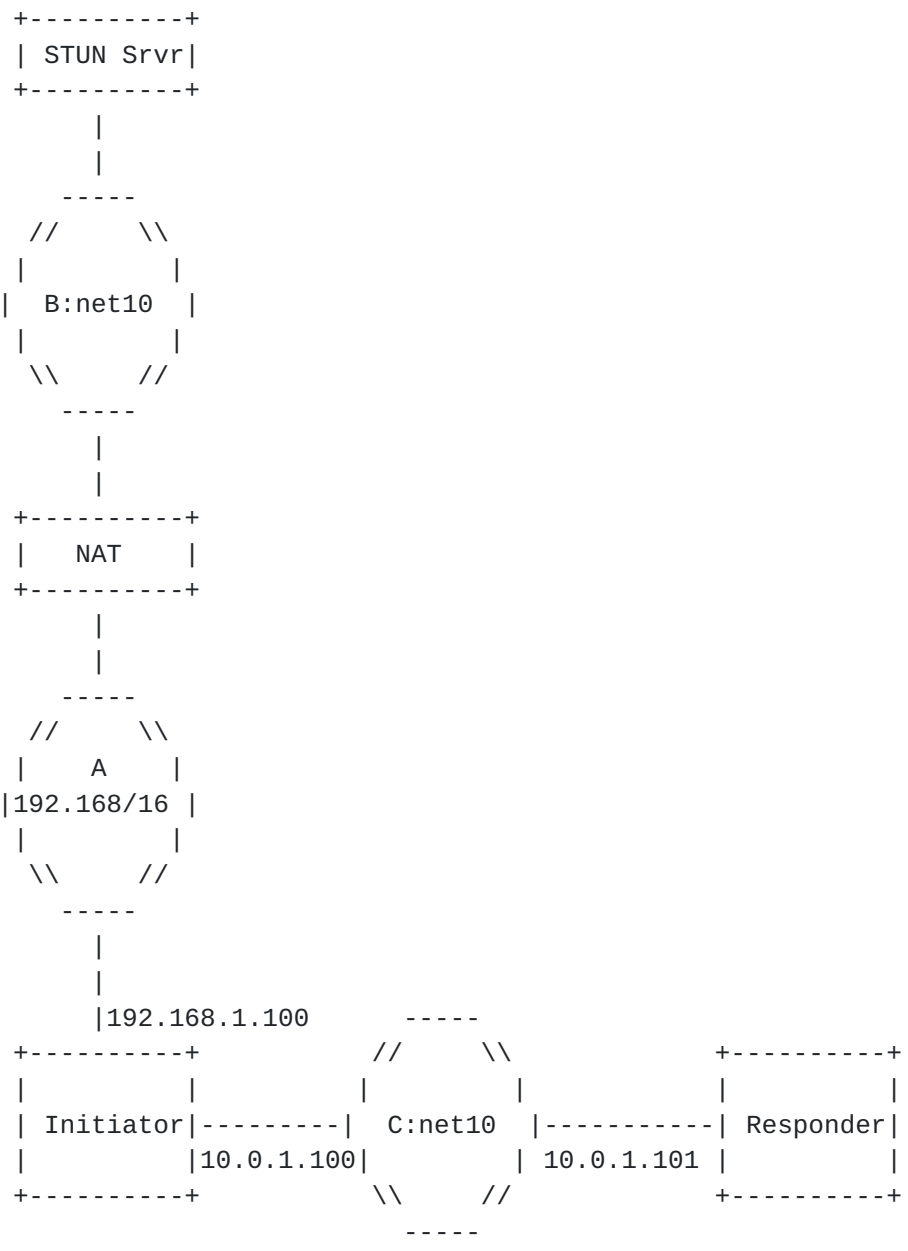


Figure 11: Identical Candidates with Different Bases

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 ICE 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 data traffic. They can then interact, through policy servers, with access routers in the network, to establish guaranteed QoS for the data 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 data, 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 ICE 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 ICE agent, a lower-priority candidate is never used until all higher-priority candidates have been tried.

B.6. Why Are Keepalives Needed?

Once data 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 data stream packets themselves (e.g., RTP) meet this objective. However, several cases merit further discussion. Firstly, in some RTP usages, such as SIP, the data 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 data in these cases. However, doing so may cause NAT bindings to timeout, and data 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 data transmission to cease sufficiently long for NAT bindings to time out.

For these reasons, the data 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 5.1.2 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 17. It is much easier for an attacker to cause an ICE 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?

Data keepalives are described in Section 11. 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 data begins flowing, network elements will assume that the data stream has a fairly regular structure, making use of periodic packets at fixed intervals, with the possibility of jitter. If an ICE agent is sending data packets, and then receives a Binding request, it would need to generate a response packet along with its data packets. This will increase the actual bandwidth requirements for the 5-tuple carrying the data 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 data.

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.

B.9. Selecting Candidate Type Preference

One criterion for selection of the type and local preference values is the use of a data intermediary, such as a TURN server, a tunnel service such as VPN server, or NAT. With a data intermediary, if data is sent to that candidate, it will first transit the data intermediary before being received. Relayed candidates are one type of candidate that involves a data intermediary. Another are host candidates obtained from a VPN interface. When data is transited through a data intermediary, it can have a positive or negative effect on the latency between transmission and reception. It may or may not increase the packet losses, because of the additional router

hops that may be taken. It may increase the cost of providing service, since data will be routed in and right back out of a data intermediary run by a provider. If these concerns are important, the type preference for relayed candidates must be carefully chosen.

Another criterion for selection of preferences is IP address family. ICE works with both IPv4 and IPv6. It 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. Implementation should follow the guidelines from [I-D.ietf-ice-dualstack-fairness] to avoid excessive delays in the connectivity check phase if broken paths exist.

Another criterion for selecting preferences is topological awareness. This is most useful for candidates that make use of intermediaries. In those cases, if an ICE 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.

Another criterion for selecting preferences might be security or privacy. 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 or similar tunnel 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.

Appendix C. Connectivity Check Bandwidth

The tables below show, for IPv4 and IPv6, the bandwidth required for performing connectivity checks, using different Ta values (given in ms) and different ufrag sizes (given in bytes).

The results were provided by Jusin Uberti (Google) 11th April 2016.

```
   IP version: IPv4
   Packet len (bytes): 108 + ufrag
        |
     ms |     4     8    12    16
   -----|------------------------
    500 | 1.86k 1.98k 2.11k 2.24k
    200 | 4.64k 4.96k 5.28k  5.6k
    100 | 9.28k 9.92k 10.6k 11.2k
     50 | 18.6k 19.8k 21.1k 22.4k
     20 | 46.4k 49.6k 52.8k 56.0k
     10 | 92.8k 99.2k  105k  112k
      5 |  185k  198k  211k  224k
      2 |  464k  496k  528k  560k
      1 |  928k  992k 1.06M 1.12M

   IP version: IPv6
   Packet len (bytes): 128 + ufrag
        |
     ms |     4     8    12    16
   -----|------------------------
    500 | 2.18k  2.3k 2.43k 2.56k
    200 | 5.44k 5.76k 6.08k  6.4k
    100 | 10.9k 11.5k 12.2k 12.8k
     50 | 21.8k 23.0k 24.3k 25.6k
     20 | 54.4k 57.6k 60.8k 64.0k
     10 |  108k  115k  121k  128k
      5 |  217k  230k  243k  256k
      2 |  544k  576k  608k  640k
      1 | 1.09M 1.15M 1.22M 1.28M
```

Figure 12: Connectivity Check Bandwidth

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