MMUSIC
Internet-Draft
Obsoletes: 5245 (if approved)
Intended status: Standards Track
Expires: June 23, 2016

A. Keranen
Ericsson
J. Rosenberg
jdrosen.net

December 21, 2015

Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal
draft-ietf-ice-rfc5245bis-01

Abstract

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

This document obsoletes RFC 5245.

Status of This Memo

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

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

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

This Internet-Draft will expire on June 23, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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

Table of Contents

1. Introduction 5
2. Overview of ICE 6
2.1. Gathering Candidate Addresses 8
2.2. Connectivity Checks 10
2.3. Sorting Candidates 11
2.4. Frozen Candidates 12
2.5. Security for Checks 13
2.6. Concluding ICE 13
2.7. Lite Implementations 15
2.8. Usages of ICE 15
3. Terminology 15
4. ICE Candidate Gathering and Exchange 19
4.1. Procedures for Full Implementation 20
4.1.1. Gathering Candidates 20
4.1.1.1. Host Candidates 21
4.1.1.2. Server Reflexive and Relayed Candidates 22
4.1.1.3. Computing Foundations 24
4.1.1.4. Keeping Candidates Alive 24
4.1.2. Prioritizing Candidates 24
4.1.2.1. Recommended Formula 25
4.1.2.2. Guidelines for Choosing Type and Local Preferences 26
4.1.3. Eliminating Redundant Candidates 27
4.2. Lite Implementation Procedures 27
4.3. Encoding the Candidate Information 28
5. ICE Candidate Processing 30
5.1. Procedures for Full Implementation 30
5.1.1. Verifying ICE Support 30
5.1.2. Determining Role 31

5.1.3. Forming the Check Lists 32
5.1.3.1. Forming Candidate Pairs 32
5.1.3.2. Computing Pair Priority and Ordering Pairs . . . 35
5.1.3.3. Pruning the Pairs 35
5.1.3.4. Computing States 35
5.1.4. Scheduling Checks 38
5.2. Lite Implementation Procedures 40
6. Performing Connectivity Checks 40
6.1. STUN Client Procedures 40
6.1.1. Creating Permissions for Relayed Candidates 40
6.1.2. Sending the Request 41
6.1.2.1. PRIORITY and USE-CANDIDATE 41
6.1.2.2. ICE-CONTROLLED and ICE-CONTROLLING 41
6.1.2.3. Forming Credentials 42
6.1.2.4. DiffServ Treatment 42
6.1.3. Processing the Response 42
6.1.3.1. Failure Cases 42
6.1.3.2. Success Cases 43
6.1.3.2.1. Discovering Peer Reflexive Candidates 43
6.1.3.2.2. Constructing a Valid Pair 44
6.1.3.2.3. Updating Pair States 45
6.1.3.2.4. Updating the Nominated Flag 46
6.1.3.3. Check List and Timer State Updates 46
6.2. STUN Server Procedures 47
6.2.1. Additional Procedures for Full Implementations . . . 47
6.2.1.1. Detecting and Repairing Role Conflicts 47
6.2.1.2. Computing Mapped Address 49
6.2.1.3. Learning Peer Reflexive Candidates 49
6.2.1.4. Triggered Checks 49
6.2.1.5. Updating the Nominated Flag 51
6.2.2. Additional Procedures for Lite Implementations . . . 51
7. Concluding ICE Processing 51
7.1. Procedures for Full Implementations 51
7.1.1. Nominating Pairs 51
7.1.1.1. Regular Nomination 52
7.1.1.2. Aggressive Nomination 52
7.1.2. Updating States 53
7.2. Procedures for Lite Implementations 54
7.2.1. Peer Is Full 55
7.2.2. Peer Is Lite 55
7.3. Freeing Candidates 56
7.3.1. Full Implementation Procedures 56
7.3.2. Lite Implementation Procedures 56
8. ICE Restarts 56
9. Keepalives 57
10. Media Handling 58
10.1. Sending Media 58
10.1.1. Procedures for Full Implementations 58

10.1.2. Procedures for Lite Implementations 59
10.1.3. Procedures for All Implementations 59
10.2. Receiving Media 59
11. Extensibility Considerations 60
12. Setting Ta and RTO 61
12.1. Real-time Media Streams 61
12.2. Non-real-time Sessions 63
13. Example 63
14. Security Considerations 68
14.1. Attacks on Connectivity Checks 68
14.2. Attacks on Server Reflexive Address Gathering 71
14.3. Attacks on Relayed Candidate Gathering 72
14.4. Insider Attacks 72
14.4.1. STUN Amplification Attack 72
15. STUN Extensions 73
15.1. New Attributes 73
15.2. New Error Response Codes 74
16. Operational Considerations 74
16.1. NAT and Firewall Types 74
16.2. Bandwidth Requirements 74
16.2.1. STUN and TURN Server Capacity Planning 74
16.2.2. Gathering and Connectivity Checks 75
16.2.3. Keepalives 75
16.3. ICE and ICE-lite 76
16.4. Troubleshooting and Performance Management 76
16.5. Endpoint Configuration 76
17. IANA Considerations 77
17.1. STUN Attributes 77
17.2. STUN Error Responses 77
18. IAB Considerations 77
18.1. Problem Definition 77
18.2. Exit Strategy 78
18.3. Brittleness Introduced by ICE 78
18.4. Requirements for a Long-Term Solution 79
18.5. Issues with Existing NAPT Boxes 80
19. Changes from RFC 5245 80
20. Acknowledgements 80
21. References 81
21.1. Normative References 81
21.2. Informative References 81
Appendix A. Lite and Full Implementations 85
Appendix B. Design Motivations 86
B.1. Pacing of STUN Transactions 86
B.2. Candidates with Multiple Bases 87
B.3. Purpose of the Related Address and Related Port Attributes 89
B.4. Importance of the STUN Username 90
B.5. The Candidate Pair Priority Formula 91

B.6. Why Are Keepalives Needed? 91
B.7. Why Prefer Peer Reflexive Candidates? 92
B.8. Why Are Binding Indications Used for Keepalives? 92
Authors' Addresses 92

1. Introduction

Protocols establishing multimedia sessions between peers typically involve exchanging IP addresses and ports for the media sources and sinks. However this poses challenges when operated through Network Address Translators (NATs) [RFC3235]. These protocols also seek to create a media flow directly between participants, so that there is no application layer intermediary between them. This is done to reduce media latency, decrease packet loss, and reduce the operational costs of deploying the application. However, this is difficult to accomplish through NAT. A full treatment of the reasons for this is beyond the scope of this specification.

Numerous solutions have been defined for allowing these protocols to operate through NAT. These include Application Layer Gateways (ALGs), the Middlebox Control Protocol [RFC3303], the original Simple Traversal of UDP Through NAT (STUN) [RFC3489] specification, and Realm Specific IP [RFC3102] [RFC3103] along with session description extensions needed to make them work, such as the Session Description Protocol (SDP) [RFC4566] attribute for the Real Time Control Protocol (RTCP) [RFC3605]. Unfortunately, these techniques all have pros and cons which, make each one optimal in some network topologies, but a poor choice in others. The result is that administrators and implementors are making assumptions about the topologies of the networks in which their solutions will be deployed. This introduces complexity and brittleness into the system. What is needed is a single solution that is flexible enough to work well in all situations.

This specification defines Interactive Connectivity Establishment (ICE) as a technique for NAT traversal for UDP-based media streams (though ICE has been extended to handle other transport protocols, such as TCP [RFC6544]). ICE works by exchanging a multiplicity of IP addresses and ports which are then tested for connectivity by peer-to-peer connectivity checks. The IP addresses and ports are exchanged via mechanisms (for example, including in a offer/answer exchange) and the connectivity checks are performed using Session Traversal Utilities for NAT (STUN) specification [RFC5389]. ICE also makes use of Traversal Using Relays around NAT (TURN) [RFC5766], an extension to STUN. Because ICE exchanges a multiplicity of IP addresses and ports for each media stream, it also allows for address selection for multihomed and dual-stack hosts, and for this reason it deprecates [RFC4091] and [RFC4092].

2. Overview of ICE

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

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

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

```
                    +---------+
   +--------+       |Signaling|         +--------+
   | STUN   |       |Server   |         | STUN   |
   | Server |       +---------+         | Server |
   +--------+      /           \        +--------+
                  /             \
                 /               \
                / <- Signaling -> \
               /                   \
         +--------+               +--------+
         |  NAT   |               |  NAT   |
         +--------+               +--------+
           /                             \
          /                               \
      +-------+                       +-------+
      | Agent |                       | Agent |
      |   L   |                       |   R   |
      +-------+                       +-------+
```

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:

- o A transport address on a directly attached network interface

- o A translated transport address on the public side of a NAT (a "server reflexive" address)

- o A transport address allocated from a TURN server (a "relayed address")

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

2.1. Gathering Candidate Addresses

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

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

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

```
                 To Internet

                     |
                     |
                     |  /------------  Relayed
                 Y:y | /               Address
                 +--------+
                 |        |
                 |  TURN  |
                 | Server |
                 |        |
                 +--------+
                     |
                     |
                     | /------------  Server
             X1':x1'|/               Reflexive
              +------------+         Address
              |    NAT     |
              +------------+
                     |
                     | /------------  Local
                 X:x |/               Address
                 +--------+
                 |        |
                 | Agent  |
                 |        |
                 +--------+
```

Figure 2: Candidate Relationships

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

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

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

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

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

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

2.2. Connectivity Checks

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

The basic principle of the connectivity checks is simple:

1. Sort the candidate pairs in priority order.

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

3. Acknowledge checks received from the other agent.

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

```
L                        R
-                        -
STUN request ->             \  L's
          <- STUN response  /  check

           <- STUN request  \  R's
STUN response ->            /  check
```

Figure 3: Basic Connectivity Check

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

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

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

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

2.3. Sorting Candidates

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

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

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

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

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

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

2.4. Frozen Candidates

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

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

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

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

2.5. Security for Checks

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

2.6. Concluding ICE

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

Consequently, ICE assigns one of the agents in the role of the CONTROLLING AGENT, and the other of the CONTROLLED AGENT. The controlling agent gets to nominate which candidate pairs will get used for media amongst the ones that are valid. It can do this in one of two ways -- using REGULAR NOMINATION or AGGRESSIVE NOMINATION.

With regular nomination, the controlling agent lets the checks continue until at least one valid candidate pair for each media

stream is found. Then, it picks amongst those that are valid, and sends a second STUN request on its NOMINATED candidate pair, but this time with a flag set to tell the peer that this pair has been nominated for use. This is shown in Figure 4.

```
L                        R
-                        -
STUN request ->             \  L's
          <- STUN response  /  check

           <- STUN request  \  R's
STUN response ->            /  check

STUN request + flag ->      \  L's
          <- STUN response  /  check
```

Figure 4: Regular Nomination

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

In aggressive nomination, the controlling agent puts the flag in every connectivity check STUN request it sends. This way, once the first check succeeds, ICE processing is complete for that media stream and the controlling agent doesn't have to send a second STUN request. The selected pair will be the highest-priority valid pair whose check succeeded. Aggressive nomination is faster than regular nomination, but gives less flexibility. Aggressive nomination is shown in Figure 5.

```
L                        R
-                        -
STUN request + flag ->      \  L's
          <- STUN response  /  check

           <- STUN request  \  R's
STUN response ->            /  check
```

Figure 5: Aggressive Nomination

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

2.7. Lite Implementations

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

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

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

2.8. Usages of ICE

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

3. Terminology

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

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

This specification makes use of the following additional terminology:

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

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

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

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

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

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

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

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

Component: A component is a piece of a media stream requiring a single transport address; a media stream may require multiple components, each of which has to work for the media stream as a whole to work. For media streams based on RTP, unless RTP and

RTCP are multiplexed in the same port, there are two components per media stream -- one for RTP, and one for RTCP.

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

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

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

Relayed Candidate: A candidate obtained by sending a TURN Allocate request from a host candidate to a TURN server. The relayed candidate is resident on the TURN server, and the TURN server relays packets back towards the agent.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Aggressive Nomination: The process of picking a valid candidate pair for media traffic by including a flag in every connectivity check STUN request, such that the first one to produce a valid candidate pair is used for media.

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

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

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

4. ICE Candidate Gathering and Exchange

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

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

```
             Initiating                      Responding
               Agent                           Agent
               (I)                             (R)
   Gather,      |                               |
   prioritize,  |                               |
   eliminate    |                               |
   redundant    |                               |
   candidates,  |                               |
   Encode       |                               |
   candidates   |                               |
                |   I's Candidate Information   |
                |------------------------------>|
                |                               | Gather,
                |                               | prioritize,
                |                               | eliminate
                |                               | redundant
                |                               | candidates,
                |                               | Encode
                |                               | candidates
                |   R's Candidate Information   |
                |<------------------------------|
                |                               |
```

Figure 6: Candidate Gathering and Exchange Sequence

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

4.1. Procedures for Full Implementation

4.1.1. Gathering Candidates

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

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

4.1.1.1. Host Candidates

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

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

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

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

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

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

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

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

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

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

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

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

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

4.1.1.2. Server Reflexive and Relayed Candidates

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

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

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

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

Every Ta milliseconds thereafter, the agent can generate another new STUN or TURN transaction. This transaction can either be a retry of a previous transaction that failed with a recoverable error (such as authentication failure), or a transaction for a new host candidate and STUN or TURN server pair. The agent SHOULD NOT generate transactions more frequently than one every Ta milliseconds. See Section 12 for guidance on how to set Ta and the STUN retransmit timer, RTO.

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

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

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

4.1.1.3. Computing Foundations

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

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

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

4.1.1.4. Keeping Candidates Alive

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

4.1.2. Prioritizing Candidates

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

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

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

4.1.2.1. Recommended Formula

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

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

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

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

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

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

4.1.2.2. Guidelines for Choosing Type and Local Preferences

One criterion for selection of the type and local preference values is the use of a media intermediary, such as a TURN server, VPN server, or NAT. With a media intermediary, if media is sent to that candidate, it will first transit the media intermediary before being received. Relayed candidates are one type of candidate that involves a media intermediary. Another are host candidates obtained from a VPN interface. When media is transited through a media intermediary, it can increase the latency between transmission and reception. It can increase the packet losses, because of the additional router hops that may be taken. It may increase the cost of providing service, since media will be routed in and right back out of a media intermediary run by a provider. If these concerns are important, the type preference for relayed candidates SHOULD be lower than host candidates. The RECOMMENDED values are 126 for host candidates, 100 for server reflexive candidates, 110 for peer reflexive candidates, and 0 for relayed candidates.

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

Another criterion for selection of preferences is IP address family. ICE works with both IPv4 and IPv6. It therefore provides a transition mechanism that allows dual-stack hosts to prefer connectivity over IPv6, but to fall back to IPv4 in case the v6 networks are disconnected (due, for example, to a failure in a 6to4 relay) [RFC3056]. It can also help with hosts that have both a native IPv6 address and a 6to4 address. In such a case, higher local preferences could be assigned to the v6 addresses, followed by the 6to4 addresses, followed by the v4 addresses. This allows a site to obtain and begin using native v6 addresses immediately, yet still fall back to 6to4 addresses when communicating with agents in other sites that do not yet have native v6 connectivity.

Another criterion for selecting preferences is security. If a user is a telecommuter, and therefore connected to a corporate network and a local home network, the user may prefer their voice traffic to be routed over the VPN in order to keep it on the corporate network when communicating within the enterprise, but use the local network when communicating with users outside of the enterprise. In such a case, a VPN address would have a higher local preference than any other address.

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

4.1.3. Eliminating Redundant Candidates

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

This process is common across the initiating and responding agents.

4.2. Lite Implementation Procedures

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

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

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

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

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

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

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

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

4.3. Encoding the Candidate Information

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

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

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

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

Foundation: A sequence of up to 32 characters.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Once an agent has sent its candidate information, that agent MUST be prepared to receive both STUN and media packets on each candidate. As discussed in Section 10.1, media packets can be sent to a candidate prior to its appearance as the default destination for media.

5. ICE Candidate Processing

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

5.1. Procedures for Full Implementation

5.1.1. Verifying ICE Support

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

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

5.1.2. Determining Role

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

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

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

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

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

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

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

5.1.3. Forming the Check Lists

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

5.1.3.1. Forming Candidate Pairs

First, the agent takes each of its candidates for a media stream (called LOCAL CANDIDATES) and pairs them with the candidates it received from its peer (called REMOTE CANDIDATES) for that media stream. In order to prevent the attacks described in Section 14.4.1, agents MAY limit the number of candidates they'll accept in an candidate exchange process. A local candidate is paired with a remote candidate if and only if the two candidates have the same component ID and have the same IP address version. It is possible that some of the local candidates won't get paired with remote candidates, and some of the remote candidates won't get paired with local candidates. This can happen if one agent doesn't include candidates for the all of the components for a media stream. If this happens, the number of components for that media stream is effectively reduced, and considered to be equal to the minimum across both agents of the maximum component ID provided by each agent across all components for the media stream.

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

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

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

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

In order to aid understanding, Figure 7 shows the relationships between several key concepts -- transport addresses, candidates, candidate pairs, and check lists, in addition to indicating the main properties of candidates and candidate pairs.

```
       +-------------------------------------------+
       |                                           |
       | +---------------------+                   |
       | |+----+ +----+ +----+ |   +Type           |
       | || IP | |Port| |Tran| |   +Priority       |
       | ||Addr| |    | |    | |   +Foundation     |
       | |+----+ +----+ +----+ |   +Component ID   |
       | |      Transport      |   +Related Address |
       | |        Addr         |                   |
       | +---------------------+   +Base           |
       |             Candidate                     |
       +-------------------------------------------+
       *                                         *
       *    ************************************
       *    *
     +-------------------------------+
    .|                               |
     | Local     Remote              |
     | +----+    +----+   +default?  |
     | |Cand|    |Cand|   +valid?    |
     | +----+    +----+   +nominated?|
     |                    +State     |
     |                               |
     |                               |
     |          Candidate Pair       |
     +-------------------------------+
     *                               *
     *                  ************
     *                  *
     +------------------+
     |  Candidate Pair  |
     +------------------+
     +------------------+
     |  Candidate Pair  |
     +------------------+
     +------------------+
     |  Candidate Pair  |
     +------------------+


            Check
            List
```

Figure 7: Conceptual Diagram of a Check List

5.1.3.2. Computing Pair Priority and Ordering Pairs

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

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

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

5.1.3.3. Pruning the Pairs

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

In addition, in order to limit the attacks described in Section 14.4.1, an agent MUST limit the total number of connectivity checks the agent performs across all check lists to a specific value, and this value MUST be configurable. A default of 100 is RECOMMENDED. This limit is enforced by discarding the lower-priority candidate pairs until there are less than 100. It is RECOMMENDED that a lower value be utilized when possible, set to the maximum number of plausible checks that might be seen in an actual deployment configuration. The requirement for configuration is meant to provide a tool for fixing this value in the field if, once deployed, it is found to be problematic.

5.1.3.4. Computing States

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

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

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

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

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

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

As ICE runs, the pairs will move between states as shown in Figure 8.

```
      +-----------+
      |           |
      |           |
      |  Frozen   |
      |           |
      |           |
      +-----------+
            |
            |unfreeze
            |
            V
      +-----------+         +-----------+
      |           |         |           |
      |           | perform |           |
      |  Waiting  |-------->|In-Progress|
      |           |         |           |
      |           |         |           |
      +-----------+         +-----------+
                                  / |
                                //  |
                              //    |
                            //      |
                           /        |
                         //         |
               failure //           |success
                     //             |
                    /               |
                  //                |
                //                  |
               //                   |
              V                     V
      +-----------+         +-----------+
      |           |         |           |
      |           |         |           |
      |   Failed  |         | Succeeded |
      |           |         |           |
      |           |         |           |
      +-----------+         +-----------+
```

Figure 8: Pair State FSM

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

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

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

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

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

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

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

Completed: In this state, ICE checks have produced nominated pairs for each component of the media stream. Consequently, ICE has succeeded and media can be sent.

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

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

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

5.1.4. Scheduling Checks

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

Once the agent has computed the check lists as described in Section 5.1.3, it sets a timer for each active check list. The timer fires every Ta*N seconds, where N is the number of active check lists (initially, there is only one active check list). Implementations MAY set the timer to fire less frequently than this. Implementations SHOULD take care to spread out these timers so that they do not fire at the same time for each media stream. Ta and the retransmit timer RTO are computed as described in Section 12. Multiplying by N allows this aggregate check throughput to be split between all active check lists. The first timer fires immediately, so that the agent performs a connectivity check the moment the candidate exchange has been done, followed by the next check Ta seconds later (since there is only one active check list).

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

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

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

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

5.2. Lite Implementation Procedures

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

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

6. Performing Connectivity Checks

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

6.1. STUN Client Procedures

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

6.1.1. Creating Permissions for Relayed Candidates

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

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

6.1.2. Sending the Request

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

ICE extends STUN by defining several new attributes, including PRIORITY, USE-CANDIDATE, ICE-CONTROLLED, and ICE-CONTROLLING. These new attributes are formally defined in Section 15.1, and their usage is described in the subsections below. These STUN extensions are applicable only to connectivity checks used for ICE.

6.1.2.1. PRIORITY and USE-CANDIDATE

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

The controlling agent MAY include the USE-CANDIDATE attribute in the Binding request. The controlled agent MUST NOT include it in its Binding request. This attribute signals that the controlling agent wishes to cease checks for this component, and use the candidate pair resulting from the check for this component. Section 7.1.1 provides guidance on determining when to include it.

6.1.2.2. ICE-CONTROLLED and ICE-CONTROLLING

The agent MUST include the ICE-CONTROLLED attribute in the request if it is in the controlled role, and MUST include the ICE-CONTROLLING attribute in the request if it is in the controlling role. The content of either attribute MUST be the tie-breaker that was determined in Section 5.1.2. These attributes are defined fully in Section 15.1.

6.1.2.3. Forming Credentials

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

6.1.2.4. DiffServ Treatment

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

6.1.3. Processing the Response

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

6.1.3.1. Failure Cases

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

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

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

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

6.1.3.2. Success Cases

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

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

6.1.3.2.1. Discovering Peer Reflexive Candidates

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

- o Its type is equal to peer reflexive.

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

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

- o Its foundation is selected as described in Section 4.1.1.3.

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

6.1.3.2.2. Constructing a Valid Pair

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

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

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

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

6.1.3.2.3. Updating Pair States

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

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

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

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

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

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

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

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

6.1.3.2.4. Updating the Nominated Flag

If the agent was a controlling agent, and it had included a USE-CANDIDATE attribute in the Binding request, the valid pair generated from that check has its nominated flag set to true. This flag indicates that this valid pair should be used for media if it is the highest-priority one amongst those whose nominated flag is set. This may conclude ICE processing for this media stream or all media streams; see Section 7.

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

6.1.3.3. Check List and Timer State Updates

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

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

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

o For each frozen check list, the agent

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

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

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

6.2. STUN Server Procedures

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

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

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

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

6.2.1. Additional Procedures for Full Implementations

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

6.2.1.1. Detecting and Repairing Role Conflicts

Normally, the rules for selection of a role in Section 5.1.2 will result in each agent selecting a different role -- one controlling and one controlled. However, in unusual call flows, typically utilizing third party call control, it is possible for both agents to select the same role. This section describes procedures for checking for this case and repairing it. These procedures apply only to

usages of ICE that require conflict resolution. The usage document MUST specify whether this mechanism is needed.

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

- o 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.

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

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

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

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

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

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

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

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

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

6.2.1.2. Computing Mapped Address

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

6.2.1.3. Learning Peer Reflexive Candidates

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

- o The priority of the candidate is set to the PRIORITY attribute from the request.

- o The type of the candidate is set to peer reflexive.

- o The foundation of the candidate is set to an arbitrary value, different from the foundation for all other remote candidates. If any subsequent candidate exchanges contain this peer reflexive candidate, it will signal the actual foundation for the candidate.

- o The component ID of this candidate is set to the component ID for the local candidate to which the request was sent.

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

6.2.1.4. Triggered Checks

Next, the agent constructs a pair whose local candidate is equal to the transport address on which the STUN request was received, and a remote candidate equal to the source transport address where the request came from (which may be the peer reflexive remote candidate that was just learned). The local candidate will either be a host candidate (for cases where the request was not received through a relay) or a relayed candidate (for cases where it is received through a relay). The local candidate can never be a server reflexive candidate. Since both candidates are known to the agent, it can

obtain their priorities and compute the candidate pair priority. This pair is then looked up in the check list. There can be one of several outcomes:

- o 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.

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

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

6.2.1.5. Updating the Nominated Flag

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

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

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

6.2.2. Additional Procedures for Lite Implementations

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

7. Concluding ICE Processing

This section describes how an agent completes ICE.

7.1. Procedures for Full Implementations

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

7.1.1. Nominating Pairs

The controlling agent nominates pairs to be selected by ICE by using one of two techniques: regular nomination or aggressive nomination. If its peer has a lite implementation, an agent MUST use a regular nomination algorithm. If its peer is using ICE options (present in an ice-options attribute from the peer) that the agent does not understand, the agent MUST use a regular nomination algorithm. If

its peer is a full implementation and isn't using any ICE options or is using ICE options understood by the agent, the agent MAY use either the aggressive or the regular nomination algorithm. However, the regular algorithm is RECOMMENDED since it provides greater stability.

7.1.1.1. Regular Nomination

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

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

Regular nomination provides the most flexibility, since the agent has control over the stopping and selection criteria for checks. The only requirement is that the agent MUST eventually pick one and only one candidate pair and generate a check for that pair with the USE-CANDIDATE attribute present. Regular nomination also improves ICE's resilience to variations in implementation (see Section 11). Regular nomination is also more stable, allowing both agents to converge on a single pair for media without any transient selections, which can happen with the aggressive algorithm. The drawback of regular nomination is that it is guaranteed to increase latencies because it requires an additional check to be done.

7.1.1.2. Aggressive Nomination

With aggressive nomination, the controlling agent includes the USE-CANDIDATE attribute in every check it sends. Once the first check for a component succeeds, it will be added to the valid list and have its nominated flag set. When all components have a nominated pair in the valid list, media can begin to flow using the highest-priority nominated pair. However, because the agent included the USE-CANDIDATE attribute in all of its checks, another check may yet

complete, causing another valid pair to have its nominated flag set. ICE always selects the highest-priority nominated candidate pair from the valid list as the one used for media. Consequently, the selected pair may actually change briefly as ICE checks complete, resulting in a set of transient selections until it stabilizes.

If certain connectivity check messages are lost, ICE agents using aggressive nomination may end up with different views on the selected candidate pair. In this case, if a security protocol that is able to authenticate the communicating parties (e.g., DTLS) is used, the controlled agent may receive valid secured traffic or handshake initialization originating from the controlling agent on a candidate pair that is different from the one the controlled agent considers as the selected pair. If this happens, the controlled agent MUST consider the pair with the secured traffic as the correct selected pair. If such security protocol is not used, both agents SHOULD continue sending connectivity check messages on the selected pair even after a pair has already been selected for use. In order to prevent the problem described here, at least one check from both agents needs to fully succeed on the selected pair.

7.1.2. Updating States

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

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

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

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

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

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

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

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

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

  * The agent MAY begin transmitting media for this media stream as described in Section 10.1.

- Once the state of each check list is Completed:

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

  * If the controlling agent is using an aggressive nomination algorithm, this may result in several updated candidate exchanges as the pairs selected for media change. An agent MAY delay sending its candidates for a brief interval (one second is RECOMMENDED) in order to allow the selected pairs to stabilize.

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

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

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

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

7.2. Procedures for Lite Implementations

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

The implementation is lite, and its peer is full.

The implementation is lite, and its peer is lite.

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

7.2.1. Peer Is Full

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

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

7.2.2. Peer Is Lite

Once the candidate exchange has completed, both agents examine their candidates and those of its peer. For each media stream, each agent pairs up its own candidates with the candidates of its peer for that media stream. Two candidates are paired up when they are for the same component, utilize the same transport protocol (UDP in this specification), and are from the same IP address family (IPv4 or IPv6).

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

- o If there is more than one pair per component:

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

  - * The agent adds the selected pair for each component to the valid list. As described in Section 10.1, this will permit

media to begin flowing. However, it is possible (and in fact likely) that both agents have chosen different pairs.

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

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

7.3. Freeing Candidates

7.3.1. Full Implementation Procedures

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

7.3.2. Lite Implementation Procedures

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

8. ICE Restarts

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

An agent MUST restart ICE for a media stream if:

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

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

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

9. Keepalives

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

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

If STUN is being used for keepalives, a STUN Binding Indication is used [RFC5389]. The Indication MUST NOT utilize any authentication mechanism. It SHOULD contain the FINGERPRINT attribute to aid in demultiplexing, but SHOULD NOT contain any other attributes. It is

used solely to keep the NAT bindings alive. The Binding Indication is sent using the same local and remote candidates that are being used for media. Though Binding Indications are used for keepalives, an agent MUST be prepared to receive a connectivity check as well. If a connectivity check is received, a response is generated as discussed in [RFC5389], but there is no impact on ICE processing otherwise.

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

10. Media Handling

10.1. Sending Media

Procedures for sending media differ for full and lite implementations.

10.1.1. Procedures for Full Implementations

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

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

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

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

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

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

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

10.1.2. Procedures for Lite Implementations

A lite implementation MUST NOT send media until it has a Valid list that contains a candidate pair for each component of that media stream. Once that happens, the agent MAY begin sending media packets. To do that, it sends media to the remote candidate in the pair (setting the destination address and port of the packet equal to that remote candidate), and will send it from the local candidate.

10.1.3. Procedures for All Implementations

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

10.2. Receiving Media

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

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

RFC 3550 [RFC3550] describes an algorithm in Section 8.2 for detecting synchronization source (SSRC) collisions and loops. These algorithms are based, in part, on seeing different source transport addresses with the same SSRC. However, when ICE is used, such

changes will sometimes occur as the media streams switch between candidates. An agent will be able to determine that a media stream is from the same peer as a consequence of the STUN exchange that proceeds media transmission. Thus, if there is a change in source transport address, but the media packets come from the same peer agent, this SHOULD NOT be treated as an SSRC collision.

11. Extensibility Considerations

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

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

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

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

12. Setting Ta and RTO

During the gathering phase of ICE (Section 4.1.1) and while ICE is performing connectivity checks (Section 6), an agent sends STUN and TURN transactions. These transactions are paced at a rate of one every Ta milliseconds, and utilize a specific RTO. This section describes how the values of Ta and RTO are computed. This computation depends on whether ICE is being used with a real-time media stream (such as RTP) or something else. When ICE is used for a stream with a known maximum bandwidth, the computation in Section 12.1 MAY be followed to rate-control the ICE exchanges. For all other streams, the computation in Section 12.2 MUST be followed.

12.1. Real-time Media Streams

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

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

For each media stream i:

```
 Ta_i = (stun_packet_size / rtp_packet_size) * rtp_ptime
```

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

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

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

```
RTO = MAX (100ms, Ta * (number of pairs))
```

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

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

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

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

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

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

12.2. Non-real-time Sessions

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

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

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

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

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

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

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

13. Example

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

```
                          +-------+
                          |STUN   |
                          |Server |
                          +-------+
                              |
                     +---------------------+
                     |                     |
                     |      Internet       |
                     |                     |
                     +---------------------+
                       |                 |
                       |                 |
                +---------+              |
                |   NAT   |              |
                +---------+              |
                     |                   |
                     |                   |
                  +-----+            +-----+
                  |  L  |            |  R  |
                  +-----+            +-----+
```

Figure 9: Example Topology

Two agents, L and R, are using ICE. Both are full-mode ICE implementations and use aggressive nomination when they are controlling. Both agents have a single IPv4 address. For agent L, it is 10.0.1.1 in private address space [RFC1918], and for agent R, 192.0.2.1 on the public Internet. Both are configured with the same STUN server (shown in this example for simplicity, although in practice the agents do not need to use the same STUN server), which is listening for STUN Binding requests at an IP address of 192.0.2.2 and port 3478. TURN servers are not used in this example. Agent L is behind a NAT, and agent R is on the public Internet. The NAT has an endpoint independent mapping property and an address dependent filtering property. The public side of the NAT has an IP address of 192.0.2.3.

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

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

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

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

```
            L             NAT           STUN             R
            |RTP STUN alloc.              |              |
            |(1) STUN Req  |              |              |
            |S=$L-PRIV-1   |              |              |
            |D=$STUN-PUB-1 |              |              |
            |------------->|              |              |
            |              |(2) STUN Req  |              |
            |              |S=$NAT-PUB-1  |              |
            |              |D=$STUN-PUB-1 |              |
            |              |------------->|              |
            |              |(3) STUN Res  |              |
            |              |S=$STUN-PUB-1 |              |
            |              |D=$NAT-PUB-1  |              |
            |              |MA=$NAT-PUB-1 |              |
            |              |<-------------|              |
            |(4) STUN Res  |              |              |
            |S=$STUN-PUB-1 |              |              |
            |D=$L-PRIV-1   |              |              |
            |MA=$NAT-PUB-1 |              |              |
            |<-------------|              |              |
            |(5) L's Candidate Information|              |
            |-------------------------------------------->|
            |              |              |              | RTP STUN
            |              |              |              | alloc.
            |              |              |(6) STUN Req  |
            |              |              |S=$R-PUB-1    |
            |              |              |D=$STUN-PUB-1 |
            |              |              |<-------------|
            |              |              |(7) STUN Res  |
            |              |              |S=$STUN-PUB-1 |
            |              |              |D=$R-PUB-1    |
            |              |              |MA=$R-PUB-1   |
            |              |              |------------->|
```

```
              |(8) R's Candidate Information|              |
              |<--------------------------------------------|
              |              |(9) Bind Req  |              |Begin
              |              |S=$R-PUB-1    |              |Connectivity
              |              |D=L-PRIV-1    |              |Checks
              |              |<----------------------------|
              |              |Dropped       |              |
              |(10) Bind Req |              |              |
              |S=$L-PRIV-1   |              |              |
              |D=$R-PUB-1    |              |              |
              |USE-CAND      |              |              |
              |------------->|              |              |
              |              |(11) Bind Req |              |
              |              |S=$NAT-PUB-1  |              |
              |              |D=$R-PUB-1    |              |
              |              |USE-CAND      |              |
              |              |----------------------------->|
              |              |(12) Bind Res |              |
              |              |S=$R-PUB-1    |              |
              |              |D=$NAT-PUB-1  |              |
              |              |MA=$NAT-PUB-1 |              |
              |              |<----------------------------|
              |(13) Bind Res |              |              |
              |S=$R-PUB-1    |              |              |
              |D=$L-PRIV-1   |              |              |
              |MA=$NAT-PUB-1 |              |              |
              |<-------------|              |              |
              |RTP flows     |              |              |
              |              |(14) Bind Req |              |
              |              |S=$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   |              |
              |              |----------------------------->|
              |              |              |              |RTP 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 for RTP.

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

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

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

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

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

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

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

14. Security Considerations

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

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

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

14.1. Attacks on Connectivity Checks

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

False Invalid: An attacker can fool a pair of agents into thinking a candidate pair is invalid, when it isn't. This can be used to

cause an agent to prefer a different candidate (such as one injected by the attacker) or to disrupt a call by forcing all candidates to fail.

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

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

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

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

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

Forcing the fake valid result works in a similar way. The agent needs to wait for the Binding request from each agent, and inject a fake success response. The attacker won't need to worry about disrupting the actual response since, if the candidate is not valid, it presumably wouldn't be received anyway. However, like the fake

invalid attack, this attack is mitigated by the STUN short-term credential mechanism in conjunction with a secure candidate exchange.

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

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

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

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

If the attacker itself is identified by the fake candidate, the attack is easier to coordinate. However, if the media path is secured (e.g., using SRTP [RFC3711]), the attacker will not be able

to play the media packets, but will only be able to discard them, effectively disabling the media stream for the call. However, this attack requires the agent to disrupt packets in order to block the connectivity check from reaching the target. In that case, if the goal is to disrupt the media stream, it’s much easier to just disrupt it with the same mechanism, rather than attack ICE.

14.2. Attacks on Server Reflexive Address Gathering

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

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

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

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

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

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

14.3. 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.

14.4. 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.

14.4.1. STUN Amplification Attack

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

It is impossible to eliminate the amplification, but the volume can be reduced through a variety of heuristics. Agents SHOULD limit the

total number of connectivity checks they perform to 100. Additionally, agents MAY limit the number of candidates they'll accept.

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

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

- o 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.

15. STUN Extensions

15.1. New Attributes

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

The PRIORITY attribute indicates the priority that is to be associated with a peer reflexive candidate, should one be discovered by this check. It is a 32-bit unsigned integer, and has an attribute value of 0x0024.

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

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

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

15.2. New Error Response Codes

This specification defines a single error response code:

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

16. Operational Considerations

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

16.1. 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.

16.2. Bandwidth Requirements

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

16.2.1. STUN and TURN Server Capacity Planning

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

require N*1.7bps. For one million users, this is 1.7 Mbps, a very small number (relatively speaking).

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

16.2.2. Gathering and Connectivity Checks

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

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

16.2.3. Keepalives

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

16.3. 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.

16.4. Troubleshooting and Performance Management

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

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

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

16.5. 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.

17. 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.

17.1. STUN Attributes

IANA has registered four STUN attributes:

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

17.2. STUN Error Responses

IANA has registered following STUN error response code:

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

18. IAB Considerations

The IAB has studied the problem of "Unilateral Self-Address Fixing", which is the general process by which a agent attempts to determine its address in another realm on the other side of a NAT through a collaborative protocol reflection mechanism [RFC3424]. ICE is an example of a protocol that performs this type of function. Interestingly, the process for ICE is not unilateral, but bilateral, and the difference has a significant impact on the issues raised by IAB. Indeed, ICE can be considered a B-SAF (Bilateral Self-Address Fixing) protocol, rather than an UNSAF protocol. Regardless, the IAB has mandated that any protocols developed for this purpose document a specific set of considerations. This section meets those requirements.

18.1. 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.

18.2. Exit Strategy

>From RFC 3424, any UNSAF proposal must provide:

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

ICE itself doesn’t easily get phased out. However, it is useful even in a globally connected Internet, to serve as a means for detecting whether a router failure has temporarily disrupted connectivity, for example. ICE also helps prevent certain security attacks that have nothing to do with NAT. However, what ICE does is help phase out other UNSAF mechanisms. ICE effectively selects amongst those mechanisms, prioritizing ones that are better, and deprioritizing ones that are worse. Local IPv6 addresses can be preferred. As NATs begin to dissipate as IPv6 is introduced, server reflexive and relayed candidates (both forms of UNSAF addresses) simply never get used, because higher-priority connectivity exists to the native host candidates. Therefore, the servers get used less and less, and can eventually be remove when their usage goes to zero.

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

18.3. Brittleness Introduced by ICE

>From RFC 3424, any UNSAF proposal must provide:

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

ICE actually removes brittleness from existing UNSAF mechanisms. In particular, classic STUN (as described in RFC 3489 [RFC3489]) has several points of brittleness. One of them is the discovery process

that requires an agent to try to classify the type of NAT it is behind. This process is error-prone. With ICE, that discovery process is simply not used. Rather than unilaterally assessing the validity of the address, its validity is dynamically determined by measuring connectivity to a peer. The process of determining connectivity is very robust.

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

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

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

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

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

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

18.4. 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.

18.5. 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.

19. Changes from RFC 5245

Following is the list of changes from RFC 5245

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

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

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

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

20. 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 Christer Holmberg, Emil Ivov, Paul Kyzivat, Pal-Erik Martinsen, Simon Perrault, Eric Rescorla, Thomas Stach, Peter Thatcher, Martin Thomson, Justin Uberti, and Suhas Nandakumar.

21. References

21.1. Normative References

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

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

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

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

21.2. Informative References

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[I-D.ietf-mmusic-ice-sip-sdp]
Petit-Huguenin, M., Keranen, A., and S. Nandakumar, "Using Interactive Connectivity Establishment (ICE) with Session Description Protocol (SDP) offer/answer and Session Initiation Protocol (SIP)", draft-ietf-mmusic-ice-sip-sdp-07 (work in progress), October 2015.

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

Appendix A. Lite and Full Implementations

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

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

Consequently, a lite implementation is only appropriate for devices that will *always* be connected to the public Internet and have a public IP address at which it can receive packets from any correspondent. ICE will not function when a lite implementation is placed behind a NAT.

ICE allows a lite implementation to have a single IPv4 host candidate and several IPv6 addresses. In that case, candidate pairs are selected by the controlling agent using a static algorithm, such as the one in RFC 6724, which is recommended by this specification. However, static mechanisms for address selection are always prone to error, since they cannot ever reflect the actual topology and can never provide actual guarantees on connectivity. They are always heuristics. Consequently, if an agent is implementing ICE just to

select between its IPv4 and IPv6 addresses, and none of its IP addresses are behind NAT, usage of full ICE is still RECOMMENDED in order to provide the most robust form of address selection possible.

It is important to note that the lite implementation was added to this specification to provide a stepping stone to full implementation. Even for devices that are always connected to the public Internet with just a single IPv4 address, a full implementation is preferable if achievable. A full implementation will reduce call setup times, since ICE's aggressive mode can be used. Full implementations also obtain the security benefits of ICE unrelated to NAT traversal; in particular, the voice hammer attack described in Section 14 is prevented only for full implementations, not lite. Finally, it is often the case that a device that finds itself with a public address today will be placed in a network tomorrow where it will be behind a NAT. It is difficult to definitively know, over the lifetime of a device or product, that it will always be used on the public Internet. Full implementation provides assurance that communications will always work.

Appendix B. Design Motivations

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

B.1. Pacing of STUN Transactions

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

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

ensures that the NAT device does not get overloaded and that traffic is kept at a reasonable rate.

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

```
            First Packets              Retransmits


                  |                        |
                  |                        |
          -------+------           -------+------
         /               \        /               \
        /                 \      /                 \

        +--+    +--+    +--+    +--+    +--+    +--+
        |A1|    |B1|    |C1|    |A2|    |B2|    |C2|
        +--+    +--+    +--+    +--+    +--+    +--+

     ---+-------+-------+-------+-------+-------+------------ Time
        0       Ta      2Ta     3Ta     4Ta     5Ta
```

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

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

B.2. Candidates with Multiple Bases

Section 4.1.3 talks about eliminating candidates that have the same transport address and base. However, candidates with the same transport addresses but different bases are not redundant. When can

an agent have two candidates that have the same IP address and port, but different bases? Consider the topology of Figure 11:

```
          +----------+
          | STUN Srvr|
          +----------+
               |
               |
             -----
           //     \\
          |         |
         |  B:net10  |
          |         |
           \\     //
             -----
               |
               |
          +----------+
          |   NAT    |
          +----------+
               |
               |
             -----
           //     \\
          |    A    |
         |192.168/16 |
          |         |
           \\     //
             -----
               |
               |
               |192.168.1.100      -----
          +----------+           //     \\             +----------+
          |          |          |         |            |          |
          | Initiator|---------|  C:net10  |-----------| Responder|
          |          |10.0.1.100|         | 10.0.1.101 |          |
          +----------+           \\     //             +----------+
                                   -----
```

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 agent can know which relayed candidate is associated with which reflexive candidate, which in turn is associated with a specific host candidate. When checks for one candidate succeed and not for others, this provides useful diagnostics on what is going on in the network.

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

B.4. Importance of the STUN Username

ICE requires the usage of message integrity with STUN using its short-term credential functionality. The actual short-term credential is formed by exchanging username fragments in the candidate exchange. The need for this mechanism goes beyond just security; it is actually required for correct operation of ICE in the first place.

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

An unfortunate consequence of the non-uniqueness of IP addresses is that, in the above example, R might not even be an ICE agent. It could be any host, and the port to which the STUN packet is directed could be any ephemeral port on that host. If there is an application listening on this socket for packets, and it is not prepared to handle malformed packets for whatever protocol is in use, the operation of that application could be affected. Fortunately, since the ports exchanged are ephemeral and usually drawn from the dynamic or registered range, the odds are good that the port is not used to run a server on host R, but rather is the agent side of some protocol. This decreases the probability of hitting an allocated port, due to the transient nature of port usage in this range. However, the possibility of a problem does exist, and network deployers should be prepared for it. Note that this is not a problem specific to ICE; stray packets can arrive at a port at any time for any type of protocol, especially ones on the public Internet. As such, this requirement is just restating a general design guideline

for Internet applications -- be prepared for unknown packets on any port.

B.5. The Candidate Pair Priority Formula

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

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

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

B.6. Why Are Keepalives Needed?

Once media begins flowing on a candidate pair, it is still necessary to keep the bindings alive at intermediate NATs for the duration of the session. Normally, the media stream packets themselves (e.g., RTP) meet this objective. However, several cases merit further discussion. Firstly, in some RTP usages, such as SIP, the media streams can be "put on hold". This is accomplished by using the SDP "sendonly" or "inactive" attributes, as defined in RFC 3264 [RFC3264]. RFC 3264 directs implementations to cease transmission of media in these cases. However, doing so may cause NAT bindings to timeout, and media won’t be able to come off hold.

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

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

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

B.7. Why Prefer Peer Reflexive Candidates?

Section 4.1.2 describes procedures for computing the priority of candidate based on its type and local preferences. That section requires that the type preference for peer reflexive candidates always be higher than server reflexive. Why is that? The reason has to do with the security considerations in Section 14. It is much easier for an attacker to cause an agent to use a false server reflexive candidate than it is for an attacker to cause an agent to use a false peer reflexive candidate. Consequently, attacks against address gathering with Binding requests are thwarted by ICE by preferring the peer reflexive candidates.

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

Media keepalives are described in Section 9. These keepalives make use of STUN when both endpoints are ICE capable. However, rather than using a Binding request transaction (which generates a response), the keepalives use an Indication. Why is that?

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

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

Authors' Addresses

Ari Keranen
Ericsson
Hirsalantie 11
02420 Jorvas
Finland

Email: ari.keranen@ericsson.com

Jonathan Rosenberg
jdrosen.net
Monmouth, NJ
US

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

ICE
Internet-Draft
Obsoletes: 5245 (if approved)
Intended status: Standards Track
Expires: September 9, 2018

A. Keranen
C. Holmberg
Ericsson
J. Rosenberg
jdrosen.net
March 8, 2018

Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal
draft-ietf-ice-rfc5245bis-20

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 September 9, 2018.

Copyright Notice

Copyright (c) 2018 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 13
  2.5. Lite Implementations 13
3. ICE Usage 13
4. Terminology 13
5. ICE Candidate Gathering and Exchange 17
  5.1. Full Implementation 17
    5.1.1. Gathering Candidates 17
      5.1.1.1. Host Candidates 18
      5.1.1.2. Server Reflexive and Relayed Candidates 19
      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 24
  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 28
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 37
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 38
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 . . . 48
8. Concluding ICE Processing 49
8.1. Procedures for Full Implementations 49
8.1.1. Nominating Pairs 49
8.1.2. Updating Check List and ICE 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
12.1.1. Procedures for Lite Implementations 55
12.2. Receiving Data 55
13. Extensibility Considerations 56
14. Setting Ta and RTO 57
14.1. General 57
14.2. Ta 57
14.3. RTO 58
15. Examples 58
15.1. Example with IPv4 Addresses 59
15.2. Example with IPv6 Addresses 65
16. STUN Extensions 69
16.1. New Attributes 69
16.2. New Error Response Codes 69
17. Operational Considerations 70
17.1. NAT and Firewall Types 70
17.2. Bandwidth Requirements 70
17.2.1. STUN and TURN Server Capacity Planning 70
17.2.2. Gathering and Connectivity Checks 71
17.2.3. Keepalives 71
17.3. ICE and ICE-lite 72
17.4. Troubleshooting and Performance Management 72
17.5. Endpoint Configuration 72
18. IAB Considerations 72
18.1. Problem Definition 73
18.2. Exit Strategy 73
18.3. Brittleness Introduced by ICE 74
18.4. Requirements for a Long-Term Solution 75
18.5. Issues with Existing NAPT Boxes 75
19. Security Considerations 75
19.1. IP Address Privacy 76
19.2. Attacks on Connectivity Checks 76
19.3. Attacks on Server Reflexive Address Gathering 79
19.4. Attacks on Relayed Candidate Gathering 80
19.5. Insider Attacks 80
19.5.1. STUN Amplification Attack 80
20. IANA Considerations 81
20.1. STUN Attributes 81
20.2. STUN Error Responses 82
20.3. ICE Options 82
21. Changes from RFC 5245 83
22. Acknowledgements 84

23. References 84
23.1. Normative References 84
23.2. Informative References 85
Appendix A. Lite and Full Implementations 89
Appendix B. Design Motivations 90
B.1. Pacing of STUN Transactions 90
B.2. Candidates with Multiple Bases 92
B.3. Purpose of the Related Address and Related Port Attributes 94
B.4. Importance of the STUN Username 94
B.5. The Candidate Pair Priority Formula 96
B.6. Why Are Keepalives Needed? 96
B.7. Why Prefer Peer Reflexive Candidates? 97
B.8. Why Are Binding Indications Used for Keepalives? 97
B.9. Selecting Candidate Type Preference 97
Appendix C. Connectivity Check Bandwidth 98
Authors' Addresses 99

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 that make each one optimal in some network topologies, but a poor choice in others. The result is that administrators and implementers are making assumptions about the topologies of the networks in which their solutions will be deployed. This introduces complexity and brittleness into the system.

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 using ICE usage-specific mechanisms (e.g., including in a offer/answer exchange) and the connectivity checks are performed using STUN [RFC5389]. ICE also makes use of Traversal Using Relays around NAT (TURN) [RFC5766], an extension to STUN. Because ICE exchanges a multiplicity of IP addresses and ports for each media stream, it also allows for address selection for multihomed and dual-stack hosts. For this reason, RFC 5245 [RFC5245] deprecated the solutions previously defined in RFC 4091 [RFC4091] and RFC 4092 [RFC4092].

Appendix B provides background information and motivations regarding the design decisions that were made when designing ICE.

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 a 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.

```
                    +---------+
   +--------+       |Signaling|         +--------+
   | STUN   |       |Server   |         | STUN   |
   | Server |       +---------+         | Server |
   +--------+      /           \        +--------+
                  /             \
                 /               \
                / <- Signaling -> \
               /                   \
         +--------+               +--------+
         |  NAT   |               |  NAT   |
         +--------+               +--------+
           /                             \
          /                               \
      +-------+                       +-------+
      | Agent |                       | Agent |
      |   L   |                       |   R   |
      +-------+                       +-------+
```

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:

- o A transport address on a directly attached network interface

- o A translated transport address on the public side of a NAT (a "server reflexive" address)

- o 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 identifies and gathers one or more 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.

The first category of candidates are those with 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.

```
                 To Internet

                     |
                     |
                     |  /------------  Relayed
                 Y:y | /               Address
                 +--------+
                 |        |
                 |  TURN  |
                 | Server |
                 |        |
                 +--------+
                     |
                     |
                     | /------------  Server
             X1':x1'|/               Reflexive
              +------------+         Address
              |    NAT     |
              +------------+
                     |
                     | /------------  Local
                 X:x |/               Address
                 +--------+
                 |        |
                 | Agent  |
                 |        |
                 +--------+
```

Figure 2: Candidate Relationships

When the agent sends a 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:

```
L                        R
-                        -
STUN request ->             \  L's
          <- STUN response  /  check

           <- STUN request  \  R's
STUN response ->            /  check
```

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". When a STUN transaction succeeds, one or more candidate pairs will become so called valid pairs, and will be added to a candidate pair list called the valid list.

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 is called a "triggered check", and accelerates the process of finding valid pairs.

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, routes without data relays or NATs) are preferred over indirect routes (routes with data relays or 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 valid pair (from the valid list) 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 pair for each component of a data stream is found and then picks a valid pair and sends a STUN request on the valid pair, using an attribute to indicate to the controlled peer that it has nominated the pair. This is shown in Figure 4.

```
L                        R
-                        -
STUN request ->             \  L's
          <- STUN response  /  check

           <- STUN request  \  R's
STUN response ->            /  check

STUN request + attribute -> \  L's
          <- STUN response  /  check
```

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 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 (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].

[RFC7825] defines an ICE usage for the Real-Time Streaming Protocol (RTSP). Note, however, that the ICE usage is based on RFC 5245.

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 need to be familiar with the terminology defined in [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 keepalives) 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 responding 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).

Related Address and Port: A transport address related to a candidate, useful for diagnostics and other purposes. If a candidate is server or peer reflexive, the related address and port is equal to the base for that server or peer reflexive candidate. If the candidate is relayed, the related address and port is equal to the mapped address in the Allocate response that provided the client with that relayed candidate. If the candidate is a host candidate, the related address and port is identical to the host candidate.

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 Pair: A candidate pair whose local candidate equals the mapped address of a successful connectivity check response, and whose remote candidate equals the destination address to which the connectivity check request was sent.

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: The process of the controlling agent indicating to the controlled agent which candidate pair the ICE agents will use for sending and receiving data. The nomination process defined in this specification was referred to "regular nomination" in RFC 5245. The nomination process that was referred to "aggressive nomination" in RFC 5245 has been deprecated in this specification.

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 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.

Timer Ta: The timer for generating new STUN or TURN transactions.

Timer RTO (Retransmission Timout): The retransmission timer for a given STUN or TURN transaction.

5. ICE Candidate Gathering and Exchange

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

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.

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 uses 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:

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

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

- o 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]).

- o 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. Similarly, when host candidates corresponding to an IPv6 address generated using a mechanism that prevents location tracking are gathered, then host candidates corresponding to IPv6 link-local addresses [RFC4291] 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. However, 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 14 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 have the same foundation when all of the following are true:

- o They have the same type (host, relayed, server reflexive, or peer reflexive).
- o Their bases have the same IP address (the ports can be different).
- o For reflexive and relayed candidates, the STUN or TURN servers used to obtain them have the same IP address (the IP address used by the agent to contact the STUN or TURN server).
- o They were obtained using the same transport protocol (TCP, UDP).

Similarly, two candidates 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 (the IP addresses used by the agent to contact the STUN or TURN server), 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 IP addresses (on new or currently used 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. Higher priority values give more priority over lower values.

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 the 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. For each IP address, independent of IP address family, there MUST be zero or one candidate. With the lite implementation, ICE cannot be used to dynamically choose amongst candidates. Therefore, including more than one candidate from a particular IP address family is NOT RECOMMENDED, since only a connectivity check can truly determine whether to use one address or the other. Instead agents that have multiple public IP addresses are RECOMMENDED to run full ICE implementations to ensure the best usage of its addresses.

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. If the formula in Section 5.1.2.1 is used to calculate the priority, the type preference value SHOULD be set to 126. If a host is v4-only, the local preference value SHOULD be set to 65535. If a host is v6 or dual-stack, the local preference value SHOULD be set to 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-only, the default candidate would typically be a globally scoped IPv6 address. Dual-stack hosts SHOULD allow configuration of whether IPv4 or IPv6 is used for the default candidate, and the configuration needs to be based on which one its administrator believes has a higher chance of success in the current network environment.

The procedures in this section are 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 might define 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 values MUST be unguessable, with at least 128 bits of random number generator output used to generate the password, and at least 24 bits output to generate the username fragment.

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 a fallback mechanism to non-ICE is supported 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 (e.g., by rewriting IP addresses in SDP). 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. The sections below describe in detail the actual procedures followed by controlling and controlled agents.

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

Both agents are full: The initiating agent that 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 that 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 (third party call control) [RFC3725] 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 accept if the peer initiates a re-determination of the roles even if the criteria for doing so are not fulfilled. This can happen if the peer is compliant with RFC 5245.

6.1.2. Forming the Check Lists

There is one check list for each data stream. To form a check list, initiating and responding ICE agents form candidate pairs, compute pair priorities, order pairs by priority, prune pairs, remove lower-priority pairs, and set 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 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 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 5 shows the properties of and relationships between transport addresses, candidates, candidate pairs, and check lists.

```
       +-------------------------------------------+
       |                                           |
       | +---------------------+                   |
       | |+----+ +----+ +----+ |   +Type           |
       | || IP | |Port| |Tran| |   +Priority       |
       | ||Addr| |    | |    | |   +Foundation     |
       | |+----+ +----+ +----+ |   +Component ID   |
       | |      Transport      |   +Related Address |
       | |        Addr         |                   |
       | +---------------------+   +Base           |
       |             Candidate                     |
       +-------------------------------------------+
       *                                         *
       *    ************************************
       *    *
     +-------------------------------+
    .|                               |
     | Local     Remote              |
     | +----+    +----+   +default?  |
     | |Cand|    |Cand|   +valid?    |
     | +----+    +----+   +nominated?|
     |                    +State     |
     |                               |
     |                               |
     |          Candidate Pair       |
     +-------------------------------+
     *                               *
     *                  ************
     *                  *
     +------------------+
     |  Candidate Pair  |
     +------------------+
     +------------------+
     |  Candidate Pair  |
     +------------------+
     +------------------+
     |  Candidate Pair  |
     +------------------+


            Check
            List
```

Figure 5: 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:

```
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 19.5.1, 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 6.

```
      +-----------+
      |           |
      |           |
      |  Frozen   |
      |           |
      |           |
      +-----------+
            |
            |unfreeze
            |
            V
      +-----------+         +-----------+
      |           |         |           |
      |           | perform |           |
      |  Waiting  |-------->|In-Progress|
      |           |         |           |
      |           |         |           |
      +-----------+         +-----------+
                                  / |
                                //  |
                              //    |
                            //      |
                           /        |
                         //         |
               failure //           |success
                     //             |
                    /               |
                  //                |
                //                  |
              //                    |
             V                      V
      +-----------+         +-----------+
      |           |         |           |
      |           |         |           |
      |   Failed  |         | Succeeded |
      |           |         |           |
      |           |         |           |
      +-----------+         +-----------+
```

Figure 6: 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 RFC 5245, 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 below 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.

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 starts a check and sends the Binding 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.

6.2. Lite Implementation Procedures

Lite implementations skip 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, if needed (it is never needed for an IPv4-only host).

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 16.1. 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 the agent is using Diffserv Codepoint markings [RFC2475] in data packets that it will send, the agent SHOULD apply the same markings to Binding requests and responses that it will send.

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 (Section 7.3.1.1), 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 MUST 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 to 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 Binding request generates a hard ICMP error, the agent SHOULD set the state of the candidate pair to Failed. Implementers need to be aware that ICMP errors can be used as a method for denial of service attacks when making a decision on how and if to process ICMP errors.

7.2.5.2.3. Timeout

If the Binding request transaction times out, the ICE agent MUST 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:

- o The Binding request generated a success response; and

- o 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:

- o The type is peer reflexive.

- o The base is the local candidate of the candidate pair from which the Binding request was sent.

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

- o 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 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 might equal the pair that generated the connectivity check, a different pair in the check list, or a pair currently not in the 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 has 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 (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 a 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 3PCC), 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:

- o 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.
- o 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.
- o 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:

- o The type is peer reflexive.
- o The priority is the value of the PRIORITY attribute in the Binding request.
- o 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.
- o 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:

- o If the pair is already on the check list:
  - * If the state of that pair is Succeeded, nothing further is done.
  - * 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 Binding requests associated with the connectivity check transaction, 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 add enqueue the pair in the triggered check list associated with the check list, and set the state of the pair to Waiting, in order to trigger a new connectivity check of the pair. Creating a new connectivity check enables validating In-Progress pairs as soon as possible, without having to wait for retransmissions of the Binding requests associated with the original connectivity check transaction.
  - * If the state of that pair is Waiting, Frozen or Failed, the agent MUST enqueue the pair in the triggered check list associated with the check list (if not already present), and set the state of the pair to Waiting, in order to trigger a new connectivity check of the pair. Note that a state change of the pair from Failed to Waiting might also trigger a state change of the associated check list.

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

- o 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:

o 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 valid pair to true.

o 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 7.2.5.3.4). The procdures for the controlled agent are described in 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 each component of 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 criteria for stopping the connectivity checks and for picking 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.

Once the controlling agent has successfully nominated a candidate pair (Section 7.2.5.3.4), the agent MUST NOT nominate another pair for same same component of the data stream within the ICE session. Doing so requires an ICE restart.

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. 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.

The usage of the ’ice2’ ice option (Section 10) by endpoints supporting this specification is supposed to prevent controlling agents implemented according to RFC 5245 from using aggressive nomination.

NOTE: In RFC 5245, usage of "aggressive nomination" allowed agents to continuously nominate pairs, before a pair was eventually selected, in order to allow sending of data on those pairs. In this specification, data can always be sent on any valid pair, without nomination. Hence, there is no longer a need for aggressive nomination.

8.1.2. Updating Check List and ICE States

For both a controlling and a controlled agent, when a candidate pair for a component of a data stream gets nominated, it might impact other pairs in the check list associated with the data stream. It might also impact the state of the check list:

- o Once a candidate pair for a component of a data stream has been nominated, and the state of the check list associated with the data stream is Running, the ICE agent MUST remove all candidate pairs for the same component from the check list and from the triggered check queue. If the state of a pair is In-Progress pair, the agent cancels the In-Progress transaction. Cancellation means that the agent will not retransmit the Binding requests associated with the connectivity check transaction, will not treat the lack of response to be a failure, but will wait the duration of the transaction timeout for a response.

- o Once candidate pairs for each component of a data stream have been nominated, and the state of the check list associated with the

data stream is Running, the ICE agent sets the state of the check list to Completed.

- o Once a candidate pair for a component of a data stream has been nominated, an agent MUST continue to respond to any Binding request it might still receive for the nominated pair, and for any remaining candidate pairs in the check list associated with the data stream. As defined in Section 7.3.1.4, as the state a pair is Succeeded, an agent will no longer generate triggered checks when receiving a Binding request for the pair.

Once the state of each check list in the check list set is Completed, the agent sets the state of the ICE session to Completed.

If the state of a check list is Failed, ICE has not been able to complete for the data stream associated with the check list. The correct behavior depends on the state of the check lists in the check list set. If the controlling agent wants to continue the session without the data stream associated with the Failed check list, and if there are still one or more check lists in Running or Completed mode, the agent can let the ICE processing continue. The agent MUST take proper actions for removing the failed data stream. If the controlling agent does not want to continue the session and MUST terminate the session. The state of the ICE session is set to Failed.

If the state of each check list in the check list set is Failed, the state of the ICE session is set to Failed. Unless the controlling agent wants to continue the session without the data streams, it MUST terminate the session.

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 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):

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

When the ICE is restarted, the candidate set for the new ICE session might include some, none, or all of the candidates used in the current ICE session.

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 RFC 5245. In addition, it will ensure that an RFC 5245-compliant peer does not use aggressive nomination either, as required by Section 14 of RFC 5245 for peers which receive unknown ICE options.

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 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 only.

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:

- o 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

- o 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.

12.1.1. 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].

12.2. 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 differ from this, e.g., by defining that data will 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.

13. 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 option concept. Each extension or change to ICE is associated with an ICE option. When an agent supports such an extension or change, it provides the ICE option to the peer agent as part of the candidate exchange.

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 nomination procedure described in Section 8 eliminates some of the need for tight coordination by delegating the selection algorithm completely to the controlling agent, and 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.

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.

14. Setting Ta and RTO

14.1. 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.

14.2. Ta

ICE agents SHOULD use a 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). See Appendix B.1 for the background of using a value of 5ms with ICE.

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

14.3. RTO

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

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

Num-Of-Cands: the number of server-reflexive and relay candidates

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

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

N: the total number of connectivity checks to be performed.

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

Num-In-Progress: the number of checks in the check list set 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.

15. Examples

This section shows two ICE examples: one using IPv4 addresses, and one using IPv6 addresses.

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 [RFC1918]. 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.

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.

15.1. Example with IPv4 Addresses

The example is using the topology shown in Figure 7.

```
                            +-------+
                            |STUN   |
                            |Server |
                            +-------+
                                |
                       +---------------------+
                       |                     |
                       |      Internet       |
                       |                     |
                       +---------------------+
                         |                 |
                         |                 |
                  +---------+              |
                  |   NAT   |              |
                  +---------+              |
                       |                   |
                       |                   |
                    +-----+             +-----+
                    |  L  |             |  R  |
                    +-----+             +-----+
```

Figure 7: Example Topology

In the example, ICE agents L and R are full ICE implementations. Both agents have a single IPv4 address. Both are configured with the same STUN server. The NAT has an endpoint independent mapping property and an address dependent filtering property. The IP addresses of the ICE agents, the STUN server and the NAT are shown below;

ENTITY	IP Address	mnemonic name
ICE Agent L:	10.0.1.1	L-PRIV-1
ICE Agent R:	192.0.2.1	R-PUB-1
STUN Server:	192.0.2.2	STUN-PUB-1
NAT (Public):	192.0.2.3	NAT-PUB-1

```
L             NAT           STUN             R
|STUN alloc.   |              |              |
|(1) STUN Req  |              |              |
|S=$L-PRIV-1   |              |              |
|D=$STUN-PUB-1 |              |              |
|------------->|              |              |
|              |(2) STUN Req  |              |
```

```
              |              |S=$NAT-PUB-1  |              |
              |              |D=$STUN-PUB-1 |              |
              |              |------------->|              |
              |              |(3) STUN Res  |              |
              |              |S=$STUN-PUB-1 |              |
              |              |D=$NAT-PUB-1  |              |
              |              |MA=$NAT-PUB-1 |              |
              |              |<-------------|              |
              |(4) STUN Res  |              |              |
              |S=$STUN-PUB-1 |              |              |
              |D=$L-PRIV-1   |              |              |
              |MA=$NAT-PUB-1 |              |              |
              |<-------------|              |              |
              |(5) L's Candidate Information|              |
              |-------------------------------------------->|
              |              |              |              | STUN
              |              |              |              | alloc.
              |              |              |(6) STUN Req  |
              |              |              |S=$R-PUB-1    |
              |              |              |D=$STUN-PUB-1 |
              |              |              |<-------------|
              |              |              |(7) STUN Res  |
              |              |              |S=$STUN-PUB-1 |
              |              |              |D=$R-PUB-1    |
              |              |              |MA=$R-PUB-1   |
              |              |              |------------->|
              |(8) R's Candidate Information|              |
              |<--------------------------------------------|
              |              |         (9) Bind Req        |Begin
              |              |         S=$R-PUB-1          |Connectivity
              |              |         D=$L-PRIV-1         |Checks
              |              |         <-------------------|
              |              |         Dropped             |
              |(10) Bind Req |              |              |
              |S=$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          |              |              |
              |==============================================>|
              |              |              |              |
              |              |(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          |              |              |
              |<==============================================|
              |              |              |              |
                                 .......
              |              |              |              |
              |(18) Bind Req |              |              |
              |S=$L-PRIV-1   |              |              |
              |D=$R-PUB-1    |              |              |
              |USE-CAND      |              |              |
              |------------->|              |              |
              |              |(19) Bind Req |              |
              |              |S=$NAT-PUB-1  |              |
              |              |D=$R-PUB-1    |              |
              |              |USE-CAND      |              |
              |              |----------------------------->|
              |              |(20) Bind Res |              |
              |              |S=$R-PUB-1    |              |
              |              |D=$NAT-PUB-1  |              |
              |              |MA=$NAT-PUB-1 |              |
              |              |<-----------------------------|
              |(21) Bind Res |              |              |
              |S=$R-PUB-1    |              |              |
              |D=$L-PRIV-1   |              |              |
              |MA=$NAT-PUB-1 |              |              |
```

```
            |<-------------|              |              |
            |              |              |              |
```

Figure 8: Example Flow

Messages 1-4: Agent L gathers a host candidate from its local IP address, and from that sends a STUN Binding request to the STUN Server. The request creates a NAT binding. The NAT public IP address of the binding becomes agent L's server reflexive candidate.

Message 5: Agent L sends its local candidate information to agent R, using the signalling protocol associated with the ICE usage.

Messages 6-7: Agent R gathers a host candidate from its local IP address, and from that sends a STUN Binding request to the STUN Server. Since agent R is not behind a NAT, R's server reflexive candidate will be identical to the host candidate.

Message 8: Agent R sends its local candidate information to agent L, using the signalling protocol associated with the ICE usage.

Since both agents are full ICE implementations, the initiating agent (agent L) becomes the controlling agent.

Agents L and R both pair up the candidates. Both agents initially have two pairs. However, agent L will prune the pair containing its server reflexive candidate, resulting in just one (L1). At agent L, this pair has a local candidate of $L_PRIV_1 and remote candidate of $R_PUB_1. At agent R, there are two pairs. The highest priority pair (R1) has a local candidate of $R_PUB_1 and remote candidate of $L_PRIV_1, and the second pair (R2) has a local candidate of $R_PUB_1 and remote candidate of $NAT_PUB_1. The pairs are shown below (the pair numbers are for reference purpose only):

ENTITY	Pairs Local	Remote	Pair #	Valid
ICE Agent L:	L_PRIV_1	R_PUB_1	L1	
ICE Agent R:	R_PUB_1	L_PRIV_1	R1	
	R_PUB_1	NAT_PUB_1	R2	

Message 9: Agent R initiates a connectivity check for pair #2. As the remote candidate of the pair is the private address of agent L, the check will not be successful, as the request cannot be routed from R to L, and will be dropped by the network.

Messages 10-13: Agent L initiates a connectivity check for pair L1. The check succeeds, and L creates a new pair (L2). The local candidate of the new pair is $NAT_PUB_1 and the remote candidate is $R_PUB_1. The pair (L2) is added to the valid list of agent L. Agent L can now send and receive data on the pair (L2) if it wishes.

ENTITY	Pairs Local	Remote	Pair #	Valid
ICE Agent L:	L_PRIV_1	R_PUB_1	L1	
	NAT_PUB_1	R_PUB_1	L2	X
ICE Agent R:	R_PUB_1	L_PRIV_1	R1	
	R_PUB_1	NAT_PUB_1	R2	

Messages 14-17: When agent R receives the Binding request from agent L (message 11) it will initiate a triggered connectivity check. The pair matches one of agent R’s existing pairs (R2). The check succeeds, and the pair (R2) is added to the valid list of agent R. Agent R can now send and receive data on the pair (R2) if it wishes.

ENTITY	Pairs Local	Remote	Pair #	Valid
ICE Agent L:	L_PRIV_1	R_PUB_1	L1	
	NAT_PUB_1	R_PUB_1	L2	X
ICE Agent R:	R_PUB_1	L_PRIV_1	R1	
	R_PUB_1	NAT_PUB_1	R2	X

Messages 18-21: At some point, the controlling agent (agent L) decides to nominate a pair (L2) in the valid list. It performs a connectivity check on the pair (L2), and includes the USE-CANDIDATE attribute in the Binding request. As the check succeeds, agent L sets the nominated flag value of the pair (L2) to ’true’. Agent R sets the nominated flag value of the matching pair (R2) to ’true’. As there are no more components associated with the stream, the

nominated pairs become the selected pairs. Consequently, processing for this stream moves into the Completed state. The ICE process also moves into the Completed state.

15.2. Example with IPv6 Addresses

The example is using the topology shown in Figure 9.

```
                          +-------+
                          |STUN   |
                          |Server |
                          +-------+
                              |
                     +----------------------+
                     |                      |
                     |      Internet        |
                     |                      |
                     +----------------------+
                       |                |
                       |                |
                       |                |
                       |                |
                       |                |
                       |                |
                       |                |
                    +-----+          +-----+
                    |  L  |          |  R  |
                    +-----+          +-----+
```

Figure 9: Example Topology

In the example, ICE agents L and R are full ICE implementations. Both agents have a single IPv6 address. Both are configured with the same STUN server. The IP addresses of the ICE agents and the STUN server are shown below;

ENTITY	IP Address	mnemonic name
ICE Agent L:	2001:db8::3	L-PUB-1
ICE Agent R:	2001:db8::5	R-PUB-1
STUN Server:	2001:db8::9	STUN-PUB-1

```
           L                           STUN             R
```

```
             |STUN alloc.                  |              |
             |(1) STUN Req                 |              |
             |S=$L-PUB-1                   |              |
             |D=$STUN-PUB-1                |              |
             |---------------------------->|              |
             |(2) STUN Res                 |              |
             | S=$STUN-PUB-1               |              |
             | D=$L-PUB-1                  |              |
             | MA=$L-PUB-1                 |              |
             |<----------------------------|              |
             |(3) L's Candidate Information|              |
             |-------------------------------------------->|
             |                             |              | STUN
             |                             |              | alloc.
             |                             |(4) STUN Req  |
             |                             |S=$R-PUB-1    |
             |                             |D=$STUN-PUB-1 |
             |                             |<-------------|
             |                             |(5) STUN Res  |
             |                             |S=$STUN-PUB-1 |
             |                             |D=$R-PUB-1    |
             |                             |MA=$R-PUB-1   |
             |                             |------------->|
             |(6) R's Candidate Information|              |
             |<--------------------------------------------|
             |(7) Bind Req                 |              |
             |S=$L-PUB-1                   |              |
             |D=$R-PUB-1                   |              |
             |-------------------------------------------->|
             |(8) Bind Res                 |              |
             |S=$R-PUB-1                   |              |
             |D=$L-PUB-1                   |              |
             |MA=$L-PUB-1                  |              |
             |<--------------------------------------------|
             |Data                         |              |
             |============================================>|
             |                             |              |
             |(9) Bind Req                 |              |
             |S=$R-PUB-1                   |              |
             |D=$L-PUB-1                   |              |
             |<--------------------------------------------|
             |(10) Bind Res                |              |
             |S=$L-PUB-1                   |              |
             |D=$R-PUB-1                   |              |
             |MA=$R-PUB-1                  |              |
             |-------------------------------------------->|
             |Data                         |              |
             |<============================================|
```

```
            |                            |             |
                               .......
            |                            |             |
            |(11) Bind Req               |             |
            |S=$L-PUB-1                  |             |
            |D=$R-PUB-1                  |             |
            |USE-CAND                    |             |
            |------------------------------------------>|
            |(12) Bind Res               |             |
            |S=$R-PUB-1                  |             |
            |D=$L-PUB-1                  |             |
            |MA=$L-PUB-1                 |             |
            |<------------------------------------------|
            |             |              |             |
```

Figure 10: Example Flow

Messages 1-2: Agent L gathers a host candidate from its local IP address, and from that sends a STUN Binding request to the STUN Server. Since agent L is not behind a NAT, L’s server reflexive candidate will be identical to the host candidate.

Message 3: Agent L sends its local candidate information to agent R, using the signalling protocol associated with the ICE usage.

Messages 4-5: Agent R gathers a host candidate from its local IP address, and from that sends a STUN Binding request to the STUN Server. Since agent R is not behind a NAT, R’s server reflexive candidate will be identical to the host candidate.

Message 6: Agent R sends its local candidate information to agent L, using the signalling protocol associated with the ICE usage.

Since both agents are full ICE implementations, the initiating agent (agent L) becomes the controlling agent.

Agents L and R both pair up the candidates. Both agents initially have one pair each. At agent L, the pair (L1) has a local candidate of $L_PUB_1 and remote candidate of $R_PUB_1. At agent R, the pair (R1) has a local candidate of $R_PUB_1 and remote candidate of $L_PUB_1. The pairs are shown below (the pair numbers are for reference purpose only):

ENTITY	Pairs Local	Remote	Pair #	Valid
ICE Agent L:	L_PUB_1	R_PUB_1	L1	
ICE Agent R:	R_PUB_1	L_PUB_1	R1	

Messages 7-8: Agent L initiates a connectivity check for pair L1. The check succeeds, and the pair (L1) is added to the valid list of agent L. Agent L can now send and receive data on the pair (L1) if it wishes.

ENTITY	Pairs Local	Remote	Pair #	Valid
ICE Agent L:	L_PUB_1	R_PUB_1	L1	X
ICE Agent R:	R_PUB_1	L_PUB_1	R1	

Messages 9-10: When agent R receives the Binding request from agent L (message 7) it will initiate a triggered connectivity check. The pair matches agent R’s existing pair (R1). The check succeeds, and the pair (R1) is added to the valid list of agent R. Agent R can now send and receive data on the pair (R1) if it wishes.

ENTITY	Pairs Local	Remote	Pair #	Valid
ICE Agent L:	L_PUB_1	R_PUB_1	L1	X
ICE Agent R:	R_PUB_1	L_PUB_1	R1	X

Messages 11-12: At some point, the controlling agent (agent L) decides to nominate a pair (L1) in the valid list. It performs a connectivity check on the pair (L1), and includes the USE-CANDIDATE attribute in the Binding request. As the check succeeds, agent L sets the nominated flag value of the pair (L1) to ’true’. Agent R sets the nominated flag value of the matching pair (R1) to ’true’. As there are no more components associated with the stream, the nominated pairs become the selected pairs. Consequently, processing

for this stream moves into the Completed state. The ICE process also moves into the Completed state.

16. STUN Extensions

16.1. New Attributes

This specification defines four 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, if one will 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 will 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. The attribute 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, unless it has received a 487 response, in which case it MUST change the number (Section 7.2.5.1). The agent MAY change the number when an ICE restart occurs.

The ICE-CONTROLLING attribute is present in a Binding request. The attribute 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. As for the ICE-CONTROLLED attribute, the number is used for solving role conflicts. An agent MUST use the same number for all Binding requests, for all streams, within an ICE session, unless it has received a 487 response, in which case it MUST change the number (Section 7.2.5.1). The agent MAY change the number when an ICE restart occurs.

16.2. New Error Response Codes

This specification defines a single error response code:

487 (Role Conflict): The Binding request contained either the ICE-CONTROLLING or ICE-CONTROLLED attribute, indicating an ICE role

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

17. Operational Considerations

This section discusses issues relevant to operators operating networks where ICE will be used by endpoints.

17.1. NAT and Firewall Types

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

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

17.2. Bandwidth Requirements

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

17.2.1. STUN and TURN Server Capacity Planning

First and foremost, ICE makes use of TURN and STUN servers, which would typically be located in 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 NATs, it is exactly zero.

The planning considerations above become more significant in multi-media scenarios (e.g., audio and video conferences), and when the numbers of participants in a session grow.

17.2.2. Gathering and Connectivity Checks

The process of gathering of candidates and performing of connectivity checks can be bandwidth intensive. ICE has been designed to pace both of these processes. The gathering phase and the connectivity check phase are meant to generate traffic at roughly the same bandwidth as the data traffic itself will consume once the ICE process conclude. 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. Once ICE has concluded, the subsequent ICE keepalives will later cause a marginal increase in the total bandwidth 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 need to ensure that their ICE implementations support the pacing feature. Though this pacing does increase call setup times, it makes ICE network friendly and easier to deploy.

17.2.3. Keepalives

STUN keepalives (in the form of STUN Binding Indications) are sent in the middle of a data session. However, they are sent only in the absence of actual data traffic. In deployments with continuous media and without utilizing Voice Activity Detection (VAD), or deployments where VAD is utilized together with short interval (max 1 second) comfort noise, the keepalives are never used and there is no increase in bandwidth usage. When VAD is being used without comfort noise, 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 do not have any real impact on capacity planning.

17.3. ICE and ICE-lite

Deployments utilizing a mix of ICE and ICE-lite interoperate with each other. They have been explicitly designed to do so.

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

17.4. Troubleshooting and Performance Management

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

ICE has built-in features to help deal with these problems. Signaling servers, typically deployed in 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.

17.5. Endpoint Configuration

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

18. IAB Considerations

The IAB has studied the problem of "Unilateral Self-Address Fixing" (UNSAF), 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 the 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.

18.1. Problem Definition

From RFC 3424, any UNSAF proposal needs to provide:

> Precise definition of a specific, limited-scope problem that is to be solved with the UNSAF proposal. A short-term fix will not be generalized in order 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.

18.2. Exit Strategy

From RFC 3424, any UNSAF proposal needs to 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. 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 removed when their usage goes to zero.

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

18.3. Brittleness Introduced by ICE

From RFC 3424, any UNSAF proposal needs to 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.

18.4. Requirements for a Long-Term Solution

From RFC 3424, any UNSAF proposal needs to 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.

18.5. Issues with Existing NAPT Boxes

From RFC 3424, any UNSAF proposal needs to 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.

19. Security Considerations

19.1. IP Address Privacy

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, ICE usages can define mechanisms for controlling which addresses are revealed to the negotiation and/or probing process. Individual implementations may also have implementation-specific rules for controlling which addresses are revealed. For example, [I-D.ietf-rtcweb-ip-handling] provides additional information about the privacy aspects of revealing IP addresses via ICE for WebRTC applications. ICE implementations where such issues can arise are RECOMMENDED to provide a programmatic or user interface that provides control over which network interfaces are used to generate candidates.

Based on the types of candidates provided by the peer, and the results of the connectivity tests performed against those candidates, the peer might be able to determine characteristics of the local network, e.g. if different timings are apparent to the peer. In the limit the peer might be able to probe the local network.

There are several types of attacks possible in an ICE system. The subsections consider these attacks and their countermeasures.

19.2. 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. Depending on the type of attack, the attacker needs to have different capabilities. In some cases the attacker needs to be on the path of the connectivity checks. In other cases the attacker does not need to be on the path, as long as it is able to generate STUN connectivity checks. While attacks on connectivity checks are typically performed by network entities, if an attacker is able to control an endpoint it might be able to trigger connectivity check attacks. 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 is not expected to. 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 does not actually route to that agent (e.g., by injecting a false peer reflexive candidate or false server reflexive candidate). The attacker then launches 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), or drop the response so that it never reaches the agent. 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. The ability for the attacker to generate a fake response is mitigated through the STUN short-term credential mechanism. 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.

Spoofed ICMP Hard Errors (Type 3, codes 2-4) can also be used to create false invalid results. If an ICE agent implements a response to these ICMP errors, and the attacker is capable of generating an ICMP message that is delivered to the agent sending the connectivity check. The validation of the ICMP error message by the agent is its only defence. For Type 3 code=4 the outer IP header provides no validation, unless the connectivity check was sent with DF=0. For code 2 or 3, which are originated by the host, the address is expected to be any of the remote agents host, reflexive, or relay candidates IP addresses. The ICMP message include the IP header and

UDP header of the message triggering the error. These fields also need to be validated. The IP destination and UDP destination port need to match either the targeted candidate address and port, or the candidate's base address. The source IP address and port can be any candidate for the same base address of the agent sending the connectivity check. Thus any attacker having access to the exchange of the candidates will have the necessary information. Thus the validation is a weak defence, and the sending of spoofed ICMP attacks is possible also for off-path attackers from a node in a network without source address validation.

Forcing the fake valid result works in a similar way. The attacker needs to wait for the Binding request from each agent, and inject a fake success response. Again, due to the STUN short-term credential mechanism, in order for the attacker to inject a valid success response, the attacker needs the password. Alternatively, the attacker can route (e.g., using a tunnelling mechanism) a valid success response, that normally would be dropped or rejected by the network, to the agent.

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 needs to 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 also needs to 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 then needs to 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 needs to intercept the check towards this false candidate, and replay it towards the other agent. Then, it needs to 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 (e.g., 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.

19.3. 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 DNSSEC 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, 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 also needs to 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.

19.4. 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.

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 needs to launch a false valid on a false candidate, per above, which is a very difficult attack to coordinate.

19.5. 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.

19.5.1. STUN Amplification Attack

The STUN amplification attack is similar to a "voice hammer" attack, where the attacker causes other agents to direct voice packets to the attack target. 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. In the case of WebRTC the user might not even be aware that this attack is ongoing, since it might be triggered in the background by malicious JavaScript code that the user has fetched. The answerer will start a new connectivity check every Ta ms (say, Ta=50ms). 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 will 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:

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

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

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

20. IANA Considerations

The original ICE specification registered four 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
```

NOTE TO IANA: Please replace the reference to RFC 5245 in the registry with a reference to this specification.

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.
```

NOTE TO IANA: Please replace the reference to RFC 5245 in the registry with a reference to this specification.

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: IESG
E-mail: iesg@ietf.org

Change control:

IESG

Description:

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

Reference:

RFC XXXX

21. Changes from RFC 5245

The purpose of this updated ICE specification is to:

- o Clarify procedures in RFC 5245.
- o Make technical changes, due to discovered flaws in RFC 5245 and based on feedback from the community that has implemented and deployed ICE applications based on RFC 5245.
- o 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:

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

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

22. 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. Ben Campbell did the AD review. Stephen Farrell did the sec-dir review. Stewart Bryant did the gen-art review. Qin We did the ops-dir review. Magnus Westerlund did the tsv-art review.

23. References

23.1. 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>.

23.2. Informative References

[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>.

[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>.

[RFC3725] Rosenberg, J., Peterson, J., Schulzrinne, H., and G. Camarillo, "Best Current Practices for Third Party Call Control (3pcc) in the Session Initiation Protocol (SIP)", BCP 85, RFC 3725, DOI 10.17487/RFC3725, April 2004, <https://www.rfc-editor.org/info/rfc3725>.

[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>.

[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>.

[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>.

[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>.

[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>.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP’s Retransmission Timer", RFC 6298, DOI 10.17487/RFC6298, June 2011, <https://www.rfc-editor.org/info/rfc6298>.

[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>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP’s Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <https://www.rfc-editor.org/info/rfc6928>.

[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>.

[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>.

[RFC7825] Goldberg, J., Westerlund, M., and T. Zeng, "A Network Address Translator (NAT) Traversal Mechanism for Media Controlled by the Real-Time Streaming Protocol (RTSP)", RFC 7825, DOI 10.17487/RFC7825, December 2016, <https://www.rfc-editor.org/info/rfc7825>.

[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-16 (work in progress), November 2017.

[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.

[I-D.ietf-rtcweb-ip-handling]
Uberti, J. and G. Shieh, "WebRTC IP Address Handling Requirements", draft-ietf-rtcweb-ip-handling-06 (work in progress), March 2018.

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, and only supports the controlled role in a session.

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. 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. Discussions in the IETF ICE WG during the work on this specification concluded that, 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 MUST 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:

```
            First Packets              Retransmits


                   |                        |
                   |                        |
             -------+------           -------+------
            /               \        /               \
           /                 \      /                 \

           +--+    +--+    +--+    +--+    +--+    +--+
           |A1|    |B1|    |C1|    |A2|    |B2|    |C2|
           +--+    +--+    +--+    +--+    +--+    +--+

        ---+-------+-------+-------+-------+-------+------------ Time
           0       Ta      2Ta     3Ta     4Ta     5Ta
```

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.

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.

- o 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, as defined in Section 2 of [RFC6928].

  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 [RFC5389] and the the TCP initial RTO, which is 1 sec in [RFC6298].

3. Let MinPacing be the minimum pacing interval between transactions, which is 5ms (see above).

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

- o 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.

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

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:

```
          +----------+
          | STUN Srvr|
          +----------+
               |
               |
             -----
           //     \\
          |         |
         |  B:net10  |
          |         |
           \\     //
             -----
               |
               |
          +----------+
          |   NAT    |
          +----------+
               |
               |
             -----
           //     \\
          |    A    |
         |192.168/16 |
          |         |
           \\     //
             -----
               |
               |
               |192.168.1.100      -----
          +----------+           //     \\             +----------+
          |          |          |         |            |          |
          | Initiator|---------|  C:net10  |-----------| Responder|
          |          |10.0.1.100|         | 10.0.1.101 |          |
          +----------+           \\     //             +----------+
                                   -----
```

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 need to 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 19. 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 Public Switched Telephone Network (PSTN) gateways and Session Border Controllers (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 needs to 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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 12, 2016

E. Ivov
Jitsi
E. Rescorla
RTFM, Inc.
J. Uberti
Google
P. Saint-Andre
&yet

December 10, 2015

Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol
draft-ietf-ice-trickle-01

Abstract

This document describes an extension to the Interactive Connectivity Establishment (ICE) protocol that enables ICE agents to send and receive candidates incrementally rather than exchanging complete lists. With such incremental provisioning, ICE agents can begin connectivity checks while they are still gathering candidates and considerably shorten the time necessary for ICE processing to complete. This mechanism is called "trickle ICE".

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 June 12, 2016.

Copyright Notice

Copyright (c) 2015 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.

Table of Contents

1. Introduction 3
2. Terminology 4
3. Determining Support for Trickle ICE 5
4. Sending the Initial Offer 6
5. Receiving the Initial Offer 6
5.1. Sending the Initial Answer 7
5.2. Forming Check Lists and Beginning Connectivity Checks 7
6. Receiving the Initial Answer 8
7. Performing Connectivity Checks 8
7.1. Scheduling Checks 8
7.2. Check List and Timer State Updates 9
8. Discovering and Sending Additional Local Candidates 9
8.1. Pairing Newly Learned Candidates and Updating Check Lists 11
8.2. Announcing End of Candidates 12
9. Receiving Additional Remote Candidates 14
10. Receiving an End-Of-Candidates Notification 14
11. Trickle ICE and Peer Reflexive Candidates 14
12. Concluding ICE Processing 15
13. Subsequent Offer/Answer Exchanges 15
14. Unilateral Use of Trickle ICE (Half Trickle) 15
15. Example Flow 16
16. IANA Considerations 17
17. Security Considerations 17
18. Acknowledgements 17
19. References 17
19.1. Normative References 17
19.2. Informative References 18
Appendix A. Interaction with ICE 19
Appendix B. Interaction with ICE Lite 20
Appendix C. Changes from Earlier Versions 21
C.1. Changes from draft-ietf-ice-trickle-00 21
C.2. Changes from draft-mmusic-trickle-ice-02 22
C.3. Changes from draft-ivov-01 and draft-mmusic-00 22
C.4. Changes from draft-ivov-00 22

C.5. Changes from draft-rescorla-01 23
C.6. Changes from draft-rescorla-00 24
Authors' Addresses 24

1. Introduction

The Interactive Connectivity Establishment (ICE) protocol [rfc5245bis] describes mechanisms for gathering candidates, prioritizing them, choosing default ones, exchanging them with the remote party, pairing them, and ordering them into check lists. Once all of these actions have been completed (and only then), the participating agents can begin a phase of connectivity checks and eventually select the pair of candidates that will be used in a media session.

Although the sequence described above has the advantage of being relatively straightforward to implement and debug once deployed, it can also be rather lengthy. Candidate gathering often involves things like querying STUN [RFC5389] servers, discovering UPnP devices, and allocating relayed candidates at TURN [RFC5766] servers. All of these actions can be delayed for a noticeable amount of time; although they can be run in parallel, they still need to respect the pacing requirements from [rfc5245bis], which is likely to delay them even further. Some or all of these actions also need be completed by the remote agent. Both agents would next perform connectivity checks and only then would they be ready to begin streaming media.

These factors can lead to relatively lengthy session establishment times and degraded user experience.

This document defines an alternative mode of operation for ICE implementations, known as "Trickle ICE", in which candidates can be exchanged incrementally. This enables ICE agents to exchange candidates as soon as a session has been initiated. Connectivity checks for a media stream can also start as soon as the first candidates for that stream become available.

Trickle ICE can reduce session establishment times in cases where connectivity is confirmed for the first exchanged candidates (e.g., where the host candidates for one of the agents are directly reachable from the second agent, such as host candidates at a media relay). Even when this is not the case, running candidate gathering for both agents and connectivity checks in parallel can considerably shorten ICE processing times.

It is worth noting that there is quite a bit of operational experience with the Trickle ICE technique, going back as far as 2005 (when the XMPP Jingle extension defined a "dribble mode" as specified

in [XEP-0176]); this document incorporates feedback from those who have implemented and deployed the technique.

In addition to the basics of Trickle ICE, this document also describes how to discover support for Trickle ICE, how regular ICE processing needs to be modified when building and updating check lists, and how Trickle ICE implementations interoperate with agents that only implement so-called "Vanilla ICE" processing as defined in [rfc5245bis].

This specification does not define the usage of Trickle ICE with any specific signalling protocol (however, see [I-D.ietf-mmusic-trickle-ice-sip] for usage with SIP [RFC3261]). Similarly, it does not define Trickle ICE in terms of the Session Description Protocol (SDP) [RFC4566] or the offer/answer model [RFC3264] because the technique can be and already is used in application protocols that are not tied to SDP or to offer/answer semantics.

2. Terminology

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

This specification makes use of all terminology defined for Interactive Connectivity Establishment in [rfc5245bis].

Vanilla ICE: The Interactive Connectivity Establishment protocol as defined in [rfc5245bis].

Candidate Harvester: A module used by an ICE agent to obtain local candidates. Candidate gatherers use different mechanisms for discovering local candidates. Some of them would typically make use of protocols such as STUN or TURN. Others may also employ techniques that are not referenced within [rfc5245bis] (e.g., UPnP based port allocation or XMPP Jingle Relay Nodes [XEP-0278]).

Trickled Candidates: Candidates that a Trickle ICE agent sends after an offer or answer but within the same context. Trickled candidates can be sent in parallel with candidate gathering and connectivity checks.

Trickling/Trickle (v.): The act of sending trickled candidates.

Half Trickle: A Trickle ICE mode of operation where the offerer gathers its first generation of candidates strictly before creating and sending the offer. Once sent, that offer can be

processed by Vanilla ICE agents and does not require support for this specification. It also allows Trickle ICE capable answerers to still gather candidates and perform connectivity checks in a non-blocking way, thus roughly offering "half" the advantages of Trickle ICE. The mechanism is mostly meant for use in cases where support for trickle ICE cannot be confirmed prior to sending a initial offer.

Full Trickle: The regular mode of operation for Trickle ICE agents, in which an initial offer can include any number of candidates (even zero candidates) and does not need to include the entire first generation of candidates as in half trickle.

3. Determining Support for Trickle ICE

Application protocols that use Trickle ICE should do one of the following:

- o Provide a way for agents to verify support of Trickle ICE prior to initiating a session (XMPP’s Service Discovery [XEP-0030] is one such mechanism).

- o Make support for Trickle ICE mandatory so that user agents can assume support.

Alternately, for cases where a protocol provides neither of the foregoing methods, agents may rely on provisioning/configuration or use the half trickle procedure described in Section 14.

Prior to sending an initial offer, agents using signaling protocols that support capabilities discovery can attempt to verify whether or not the remote party supports Trickle ICE. If an agent determines that the remote party does not support Trickle ICE, it MUST fall back to using Vanilla ICE or abandon the entire session.

In application protocols that use SDP, a user agent supporting Trickle ICE MUST include a token of "trickle" in the ice-options attribute every time it generates an offer or an answer. This enables an agent that receives offers or answers to verify support by checking for presence of the token.

Dedicated discovery semantics and half trickle are needed only prior to session initiation (e.g., when sending the initial offer). After a session is established and Trickle ICE support is confirmed for both parties, either agent can use full trickle for subsequent offers.

4. Sending the Initial Offer

An agent starts gathering candidates as soon as it has an indication that communication is imminent (e.g., a user interface cue or an explicit request to initiate a session). Contrary to Vanilla ICE, implementations of Trickle ICE do not need to gather candidates in a blocking manner. Therefore, unless half trickle is being used, agents SHOULD generate and transmit their initial offer as early as possible, in order to allow the remote party to start gathering and trickling candidates.

Trickle ICE agents MAY include any set of candidates in an offer. This includes the possibility of sending an offer that contains all the candidates that the agent plans to use (as in half trickle mode), sending an offer that contains only a publically-reachable IP address (e.g., a host candidate at a media relay that is known to not be behind a firewall), or sending an offer with no candidates at all (in which case the offerer can receive the answerer’s initial candidate list sooner and the answerer can begin candidate gathering more quickly).

For optimal performance, it is RECOMMENDED that the candidates in an initial offer (if any) be host candidates only. This would allow both agents to start gathering server reflexive, relayed, and other non-host candidates simultaneously, and it would also enable them to begin connectivity checks.

If the privacy implications of revealing host addresses on an endpoint device are a concern, agents can generate an offer that contains no candidates and then only trickle candidates that do not reveal host addresses (e.g., relayed candidates).

Methods for calculating priorities and foundations, as well as determining redundancy of candidates, work just as with vanilla ICE.

5. Receiving the Initial Offer

When an agent receives an initial offer, it will first check if the offer or offerer indicates support for Trickle ICE as explained in Section 3. If this is not the case, the agent MUST process the offer according to Vanilla ICE procedures [rfc5245bis] or offer/answer processing rules [RFC3264] if no ICE support is detected at all.

If support for Trickle ICE is confirmed, an agent will automatically assume support for Vanilla ICE as well even if the support verification procedure in [rfc5245bis] indicates otherwise. Specifically, the rules from [rfc5245bis] would imply that ICE itself is not supported if the initial offer includes no candidates in the

offer; however, such a conclusion is not warranted if the answerer can confirm that the offerer supports Trickle ICE and thus fallback to [RFC3264] is not necessary.

If the offer does indicate support for Trickle ICE, the agent will determine its role, start gathering and prioritizing candidates and while doing so it will also respond by sending its own answer, so that both agents can start forming check lists and begin connectivity checks.

5.1. Sending the Initial Answer

An agent can respond to an initial offer at any point while gathering candidates. The answer can again contain any set of candidates, including all candidates or no candidates. (The benefit of including no candidates is to send the answer as quickly as possible, so that both parties can consider the overall session to be under active negotiation as soon as possible.) Unless the answering agent is protecting host addresses for privacy reasons, it would typically construct this initial answer including only host addresses, thus enabling the remote party to also start forming check lists and performing connectivity checks.

In application protocols that use SDP, the answer MUST indicate support for Trickle ICE as described in Section 3.

5.2. Forming Check Lists and Beginning Connectivity Checks

After exchanging the offer and answer, and as soon as they have obtained local and remote candidates, agents begin forming candidate pairs, computing candidate pair priorities and ordering candidate pairs, pruning duplicate pairs, and creating check lists according to the Vanilla ICE procedures described in [rfc5245bis].

According to those procedures, in order for candidate pairing to be possible and for duplicate candidates to be pruned, the candidates would need to be provided in both the offer and the answer. Under Trickle ICE, check lists can be empty until candidate pairs are sent or received. Therefore Trickle ICE agents handle check lists and candidate pairing in a slightly different way: the agents still create the check lists, but they only populate the check lists after they actually have the candidate pairs.

> Note: According to [rfc5245bis], "A check list with at least one pair that is Waiting is called an active check list, and a check list with all pairs Frozen is called a frozen check list." Formally speaking an active check list does not have a state of Active and a frozen check list does not have a state of Frozen,

because the only check list states are Running, Completed, and Failed.

A Trickle ICE agent MUST initially consider all check lists to be frozen. It then inspects the first check list and attempts to unfreeze all candidates belonging to the first component on the first media stream (i.e., the first media stream that was reported to the ICE implementation from the using application). However, if this check list is still empty, an agent delays further processing until the check list is non-empty.

With regard to pruning of duplicate candidate pairs, a Trickle ICE agent SHOULD follow a policy of "first one wins" and not re-apply the pruning procedure if a higher-priority candidate pair is received from the remote agent.

Respecting the order in which check lists have been reported to an ICE implementation is crucial to the frozen candidates algorithm, so that connectivity checks are performed simultaneously by both agents.

6. Receiving the Initial Answer

When receiving an answer, agents follow Vanilla ICE procedures to determine their role, after which they form check lists (as described in Section 5.2) and begin connectivity checks.

7. Performing Connectivity Checks

For the most part, Trickle ICE agents perform connectivity checks following Vanilla ICE procedures. However, the asynchronous nature of gathering and communicating candidates in Trickle ICE impose a number of changes described as described in the following sections.

7.1. Scheduling Checks

The ICE specification [rfc5245bis], Section 5.8, requires that agents terminate the timer for a triggered check in relation to an active check list once the agent has exhausted all frozen pairs in check list. This will not work with Trickle ICE, because more pairs will be added to the check list incrementally.

Therefore, a Trickle ICE agent SHOULD NOT terminate the timer until the state of the check list is Completed or Failed as specified herein (see Section 8.2).

7.2. Check List and Timer State Updates

The ICE specification [rfc5245bis], Section 7.1.3.3, requires that agents update check lists and timer states upon completing a connectivity check transaction. During such an update, Vanilla ICE agents would set the state of a check list to Failed if both of the following two conditions are satisfied:

- o all of the pairs in the check list are either in the Failed or Succeeded state; and

- o there is not a pair in the valid list for each component of the media stream.

With Trickle ICE, the above situation would often occur when candidate gathering and trickling are still in progress, even though it is quite possible that future checks will succeed. For this reason, Trickle ICE agents add the following conditions to the above list:

- o all candidate gatherers have completed and the agent is not expecting to discover any new local candidates;

- o the remote agent has sent an end-of-candidates indication for that check list as described in Section 8.2.

Vanilla ICE requires that agents then update all other check lists, placing one pair from each of them into the Waiting state, effectively unfreezing all remaining check lists. However, under Trickle ICE other check lists might still be empty at that point. Therefore a Trickle ICE agent SHOULD monitor whether a check list is active or frozen independently of the state of the candidate pairs that the check list contains. A Trickle ICE agent SHOULD consider a check list to be active either when unfreezing the first candidate pair in the check list or when there is no candidate pair in the check list (i.e., when the check list is empty).

8. Discovering and Sending Additional Local Candidates

After an offer or an answer has been sent, agents will most likely continue discovering new local candidates as STUN, TURN, and other non-host candidate gathering mechanisms begin to yield results. Whenever an agent discovers such a new candidate it will compute its priority, type, foundation and component ID according to normal Vanilla ICE procedures.

The new candidate is then checked for redundancy against the existing list of local candidates. If its transport address and base match

those of an existing candidate, it will be considered redundant and will be ignored. This would often happen for server reflexive candidates that match the host addresses they were obtained from (e.g., when the latter are public IPv4 addresses). Contrary to Vanilla ICE, Trickle ICE agents will consider the new candidate redundant regardless of its priority.

Next the agent sends (i.e., trickles) the newly discovered candidate(s) to the remote agent. The actual delivery of the new candidates are specified by using protocols such as SIP or XMPP. Trickle ICE imposes no restrictions on the way this is done or whether it is done at all. For example, some applications may choose not to send trickle updates for server reflexive candidates and rely on the discovery of peer reflexive ones instead.

When trickle updates are sent, each candidate MUST be delivered to the receiving Trickle ICE implementation not more than once and in the same order that they were sent. In other words, if there are any candidate retransmissions, they must be hidden from the ICE implementation.

Also, candidate trickling needs to be correlated to a specific ICE negotiation session, so that if there is an ICE restart, any delayed updates for a previous session can be recognized as such and ignored by the receiving party.

One important aspect of Vanilla ICE is that connectivity checks for a specific foundation and component are attempted simultaneously by both agents, so that any firewalls or NATs fronting the agents would whitelist both endpoints and allow all except for the first ("suicide") packets to go through. This is also crucial to unfreezing candidates in the right time.

In order to preserve this feature in Trickle ICE, when trickling candidates agents MUST respect the order of the components as they appear (implicitly or explicitly) in the offer/answer descriptions. Therefore a candidate for a specific component MUST NOT be sent prior to candidates for other components within the same foundation.

For example, the following SDP description contains two components (RTP and RTCP) and two foundations (host and server reflexive):

```
v=0
o=jdoe 2890844526 2890842807 IN IP4 10.0.1.1
s=
c=IN IP4 10.0.1.1
t=0 0
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
m=audio 5000 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=candidate:1 1 UDP 2130706431 10.0.1.1 5000 typ host
a=candidate:1 2 UDP 2130706431 10.0.1.1 5001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.3 5000 typ srflx
    raddr 10.0.1.1 rport 8998
a=candidate:2 2 UDP 1694498815 192.0.2.3 5001 typ srflx
    raddr 10.0.1.1 rport 8998
```

For this description the RTCP host candidate MUST NOT be sent prior to the RTP host candidate. Similarly the RTP server reflexive candidate MUST be sent together with or prior to the RTCP server reflexive candidate.

Note that the order restriction only applies among candidates that belong to the same foundation.

It is also equally important to preserve this order across media streams, which is covered by the requirement to always start unfreezing candidates starting from the first media stream as described under Section 5.2.

Once the candidate has been sent to the remote party, the agent checks if any remote candidates are currently known for this same stream. If not, the new candidate will simply be added to the list of local candidates.

Otherwise, if the agent has already learned of one or more remote candidates for this stream and component, it will begin pairing the new local candidates with them and adding the pairs to the existing check lists according to their priority.

8.1. Pairing Newly Learned Candidates and Updating Check Lists

Forming candidate pairs works the way it is described by the ICE specification [rfc5245bis]. However, actually adding the new pair to a check list happens according to the rules described below.

If the check list where the pair is to be added already contains the maximum number of candidate pairs (100 by default as per [rfc5245bis]), the new pair is discarded.

If the new pair's local candidate is server reflexive, the server reflexive candidate MUST be replaced by its base before adding the pair to the list. Once this is done, the agent examines the check list looking for another pair that would be redundant with the new one. If such a pair exists, the newly formed pair is ignored.

For all other pairs, including those with a server reflexive local candidate that were not found to be redundant:

- o if this check list is frozen then the new pair will be assigned a state of Frozen.

- o else if the check list is active and it is either empty or contains only candidates in the Succeeded and Failed states, then the new pair's state is set to Waiting.

- o else if the check list is non-empty and active, then the state of the new pair will be set to

  Frozen: if there is at least one pair in the check list whose foundation matches the one in the new pair and whose state is neither Succeeded nor Failed (eventually the new pair will get unfrozen after the ongoing check for the existing pair concludes);

  Waiting: if the list contains no pairs with the same foundation as the new one, or, in case such pairs exist but they are all in either the Succeeded or Failed states.

8.2. Announcing End of Candidates

Once all candidate gathering is completed or expires for a specific media stream, the agents will generate an "end-of-candidates" indication for that stream and send it to the remote agent via the signalling channel. The exact form of the indication depends on the application protocol. The indication can be sent in the following ways:

- o As part of an offer (which would typically be the case with half trickle initial offers)

- o Along with the last candidate an agent can send for a stream

- o As a standalone notification (e.g., after STUN Binding requests or TURN Allocate requests to a server timeout and the agent has no other active gatherers)

A controlled Trickle ICE agent SHOULD send end-of-candidates indications after gathering for a media stream has completed, unless ICE processing terminates before the agent has had a chance to do so. Sending the indication is necessary in order to avoid ambiguities and speed up the conclusion of ICE processing. On the other hand, a controlling agent MAY conclude ICE processing prior to sending end-of-candidates indications for all streams. This would typically be the case with aggressive nomination. However, it is RECOMMENDED that controlling agents do send such indications whenever possible for the sake of consistency and to keep middle boxes and controlled agents up-to-date on the state of ICE processing.

When sending an end-of-candidate indication during trickling (rather than as a part of an offer or an answer), it is the responsibility of the using protocol to define methods for relating the indication to one or more specific media streams.

Receiving an end-of-candidates indication enables an agent to update check list states and, in case valid pairs do not exist for every component in every media stream, determine that ICE processing has failed. It also enables agents to speed up the conclusion of ICE processing when a candidate pair has been validated but it involves the use of lower-preference transports such as TURN. In such situations, an implementations may choose to wait and see if higher-priority candidates are received; in this case the end-of-candidates indication provides a notificaiton that such candidates are not forthcoming.

An agent MAY also choose to generate an end-of-candidates indication before candidate gathering has actually completed, if the agent determines that gathering has continued for more than an acceptable period of time. However, an agent MUST NOT send any more candidates after it has send an end-of-candidates indication.

When performing half trickle, an agent SHOULD send an end-of-candidates indication together with its initial offer unless it is planning to potentially send additional candidates (e.g., in case the remote party turns out to support Trickle ICE).

When an end-of-candidates indication is sent as part of an offer or an answer, it can be considered to apply to the session as a whole, which is equivalent to having it apply to all media streams.

After an agent sends the end-of-candidates indication, it will update the state of the corresponding check list as explained in Section 7.2. Past that point, an agent MUST NOT send any new candidates within this ICE session. After an agent has received an end-of-candidates indication, it MUST also ignore any newly received candidates for that media stream or media session. Therefore, adding new candidates to the negotiation is possible only through an ICE restart.

This specification does not override Vanilla ICE semantics for concluding ICE processing. Therefore even if end-of-candidates indications are sent agents will still have to go through pair nomination. Also, if pairs have been nominated for components and media streams, ICE processing will still conclude even if end-of-candidate indications have not been received for all streams.

9. Receiving Additional Remote Candidates

At any point of ICE processing, a Trickle ICE agent may receive new candidates from the remote agent. When this happens and no local candidates are currently known for this same stream, the new remote candidates are simply added to the list of remote candidates.

Otherwise, the new candidates are used for forming candidate pairs with the pool of local candidates and they are added to the local check lists as described in Section 8.1.

Once the remote agent has completed candidate gathering, it will send an end-of-candidates indication. Upon receiving such an indication, the local agent MUST update check list states as per Section 7.2. This may lead to some check lists being marked as Failed.

10. Receiving an End-Of-Candidates Notification

When an agent receives an end-of-candidates indication for a specific check list, it will update the state of the check list as per Section 7.2. If the check list is still active state after the update, the agent will persist the the fact that an end-of-candidates indication has been received and take it into account in future updates to the check list.

11. Trickle ICE and Peer Reflexive Candidates

Even though Trickle ICE does not explicitly modify the procedures for handling peer reflexive candidates, their processing could be impacted in implementations. With Trickle ICE, it is possible that server reflexive candidates can be discovered as peer reflexive in

cases where incoming connectivity checks are received from these candidates before the trickle updates that carry them.

While this would certainly increase the number of cases where ICE processing nominates and selects candidates discovered as peer-reflexive, it does not require any change in processing.

It is also likely that some applications would prefer not to trickle server reflexive candidates to entities that are known to be publicly accessible and where sending a direct STUN binding request is likely to reach the destination faster than the trickle update that travels through the signalling path.

12. Concluding ICE Processing

This specification does not directly modify the procedures ending ICE processing described in Section 8 of [rfc5245bis], and Trickle ICE implementations will follow the same rules.

13. Subsequent Offer/Answer Exchanges

Either agent MAY generate a subsequent offer at any time allowed by [RFC3264]. When this happens agents will use [rfc5245bis] semantics to determine whether or not the new offer requires an ICE restart. If this is the case then agents would perform Trickle ICE as they would in an initial offer/answer exchange.

The only differences between an ICE restart and a brand new media session are that:

- o during the restart, media can continue to be sent to the previously validated pair.

- o both agents are already aware whether or not their peer supports Trickle ICE, and there is no longer need for performing half trickle or confirming support with other mechanisms.

14. Unilateral Use of Trickle ICE (Half Trickle)

In half trickle mode, the offerer sends a regular, Vanilla ICE offer, with a complete set of candidates. This ensures that the offer can be processed by a Vanilla ICE answerer and is mostly meant for use in cases where support for Trickle ICE cannot be confirmed prior to sending an initial offer. The initial offer indicates support for Trickle ICE, so that the answerer can respond with an incomplete set of candidates and continue trickling the rest. Half trickle offers typically contain an end-of-candidates indication, although this is not mandatory because if trickle support is confirmed then the

offerer can choose to trickle additional candidates before it sends an end-of-candidates indication.

The half trickle mechanism can be used in cases where there is no way for an agent to verify in advance whether a remote party supports Trickle ICE. Because the initial offer contains a full set of candidates, it can thus be handled by a regular Vanilla ICE agent, while still allowing a Trickle ICE agent to use the optimization defined in this specification. This prevents negotiation from failing in the former case while still giving roughly half the Trickle ICE benefits in the latter (hence the name of the mechanism).

Use of half trickle is only necessary during an initial offer/answer exchange. After both parties have received a session description from their peer, they can each reliably determine Trickle ICE support and use it for all subsequent offer/answer exchanges.

In some instances, using half trickle might bring more than just half the improvement in terms of user experience. This can happen when an agent starts gathering candidates upon user interface cues that the user will soon be initiating an offer, such as activity on a keypad or the phone going off hook. This would mean that some or all of the candidate gathering could be completed before the agent actually needs to send the offer. Because the answerer will be able to trickle candidates, both agents will be able to start connectivity checks and complete ICE processing earlier than with Vanilla ICE and potentially even as early as with full trickle.

However, such anticipation is not always possible. For example, a multipurpose user agent or a WebRTC web page where communication is a non-central feature (e.g., calling a support line in case of a problem with the main features) would not necessarily have a way of distinguishing between call intentions and other user activity. In such cases, using full trickle is most likely to result in an ideal user experience. Even so, using half trickle would be an improvement over vanilla ICE because it would result in a better experience for answerers.

15. Example Flow

A typical successful Trickle ICE exchange with an Offer/Answer protocol would look this way:

```
           Alice                                           Bob
             |                     Offer                     |
             |-------------------------------------------->|
             |            Additional Candidates              |
             |-------------------------------------------->|
             |                                               |
             |                     Answer                    |
             |<--------------------------------------------|
             |            Additional Candidates              |
             |<--------------------------------------------|
             |                                               |
             | Additional Candidates and Connectivity Checks |
             |<------------------------------------------->|
             |                                               |
             |<=============== MEDIA FLOWS =================>|
```

Figure 1: Example

16. IANA Considerations

This specification requests no actions from IANA.

17. Security Considerations

This specification inherits most of its semantics from [rfc5245bis] and as a result all security considerations described there remain the same.

18. Acknowledgements

The authors would like to thank Bernard Aboba, Flemming Andreasen, Rajmohan Banavi, Christer Holmberg, Jonathan Lennox, Enrico Marocco, Pal Martinsen, Martin Thomson, Dale R. Worley, and Brandon Williams for their reviews and suggestions on improving this document.

19. References

19.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[rfc5245bis] Keranen, A. and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", draft-ietf-ice-rfc5245bis-00 (work in progress), October 2015.

19.2. Informative References

[I-D.ietf-mmusic-trickle-ice-sip] Ivov, E., Thomas, T., Marocco, E., and C. Holmberg, "A Session Initiation Protocol (SIP) usage for Trickle ICE", draft-ietf-mmusic-trickle-ice-sip-03 (work in progress), October 2015.

[I-D.keranen-mmusic-ice-address-selection] Keraenen, A. and J. Arkko, "Update on Candidate Address Selection for Interactive Connectivity Establishment (ICE)", draft-keranen-mmusic-ice-address-selection-01 (work in progress), July 2012.

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

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

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

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

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[XEP-0030]
Hildebrand, J., Millard, P., Eatmon, R., and P. Saint-Andre, "XEP-0030: Service Discovery", XEP XEP-0030, June 2008.

[XEP-0176]
Beda, J., Ludwig, S., Saint-Andre, P., Hildebrand, J., Egan, S., and R. McQueen, "XEP-0176: Jingle ICE-UDP Transport Method", XEP XEP-0176, June 2009.

[XEP-0278]
Camargo, T., "XEP-0278: Jingle Relay Nodes", XEP XEP-0278, June 2011.

Appendix A. Interaction with ICE

The ICE protocol was designed to be flexible enough to work in and adapt to as many network environments as possible. Despite that flexibility, ICE as specified in [rfc5245bis] does not by itself support trickle ICE. This section describes how trickling of candidates interacts with ICE.

[rfc5245bis] describes the conditions required to update check lists and timer states while an ICE agent is in the Running state. These conditions are verified upon transaction completion and one of them stipulates that:

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

This could be a problem and cause ICE processing to fail prematurely in a number of scenarios. Consider the following case:

1. Alice and Bob are both located in different networks with Network Address Translation (NAT). Alice and Bob themselves have different address but both networks use the same [RFC1918] block.

2. Alice sends Bob the candidate 10.0.0.10 which also happens to correspond to an existing host on Bob’s network.

3. Bob creates a check list consisting solely of 10.0.0.10 and starts checks.

4. These checks reach the host at 10.0.0.10 in Bob’s network, which responds with an ICMP "port unreachable" error and per [rfc5245bis] Bob marks the transaction as Failed.

At this point the check list only contains Failed candidates and the valid list is empty. This causes the media stream and potentially all ICE processing to Fail.

A similar race condition would occur if the initial offer from Alice only contains candidates that can be determined as unreachable (per [I-D.keranen-mmusic-ice-address-selection]) from any of the candidates that Bob has gathered. This would be the case if Bob's candidates only contain IPv4 addresses and the first candidate that he receives from Alice is an IPv6 one.

Another potential problem could arise when a non-trickle ICE implementation sends an offer to a trickle one. Consider the following case:

1. Alice's client has a non-Trickle ICE implementation

2. Bob's client has support for Trickle ICE.

3. Alice and Bob are behind NATs with address-dependent filtering [RFC4787].

4. Bob has two STUN servers but one of them is currently unreachable

After Bob's agent receives Alice's offer it would immediately start connectivity checks. It would also start gathering candidates, which would take a long time because of the unreachable STUN server. By the time Bob's answer is ready and sent to Alice, Bob's connectivity checks may well have failed: until Alice gets Bob's answer, she won't be able to start connectivity checks and punch holes in her NAT. The NAT would hence be filtering Bob's checks as originating from an unknown endpoint.

Appendix B. Interaction with ICE Lite

The behavior of ICE lite agents that are capable of Trickle ICE does not require any particular rules other than those already defined in this specification and [rfc5245bis]. This section is hence provided only for informational purposes.

Such an agent would generate offers or answers as per [rfc5245bis]. Both its offers and answers will indicate support for Trickle ICE. Given that they will contain a complete set of candidates (the agent's host candidates), these offers and answers would also be accompanied with an end-of-candidates indication.

When performing full trickle, a full ICE implementation could send an offer or an answer with no candidates. After receiving an answer

that identifies the remote agent as an ICE lite implementation, the offerer may choose to not send any additional candidates. The same is also true in the case when the ICE lite agent is making the offer and the full ICE one is answering. In these cases the connectivity checks would be enough for the ICE lite implementation to discover all potentially useful candidates as peer reflexive. The following example illustrates one such ICE session using SDP syntax:

```
         ICE Lite                                          Bob
          Agent
            |   Offer (a=ice-lite a=ice-options:trickle)    |
            |----------------------------------------------->|
            |                                                |no cand
            |         Answer (a=ice-options:trickle)         |trickling
            |<-----------------------------------------------|
            |              Connectivity Checks               |
            |<----------------------------------------------->|
     peer rflx|                                              |
    cand disco|                                              |
            |                                                |
            |<=============== MEDIA FLOWS =================>|
```

Figure 2: Example

In addition to reducing signaling traffic this approach also removes the need to discover STUN bindings, or to make TURN or UPnP allocations, which may considerably lighten ICE processing.

Appendix C. Changes from Earlier Versions

Note to the RFC-Editor: please remove this section prior to publication as an RFC.

C.1. Changes from draft-ietf-ice-trickle-00

- o Removed dependency on SDP (which is to be provided in a separate specification).
- o Clarified text about the fact that a check list can be empty if no candidates have been sent or received yet.
- o Clarified wording about check list states so as not to define new states for "Active" and "Frozen" because those states are not defined for check lists (only for candidate pairs) in ICE core.

- o Removed open issues list because it was out of date.

- o Completed a thorough copy edit.

C.2. Changes from draft-mmusic-trickle-ice-02

- o Addressed feedback from Rajmohan Banavi and Brandon Williams.

- o Clarified text about determining support and about how to proceed if it can be determined that the answering agent does not support Trickle ICE.

- o Clarified text about check list and timer updates.

- o Clarified when it is appropriate to use half trickle or to send no candidates in an offer or answer.

- o Updated the list of open issues.

C.3. Changes from draft-ivov-01 and draft-mmusic-00

- o Added a requirement to trickle candidates by order of components to avoid deadlocks in the unfreezing algorithm.

- o Added an informative note on peer-reflexive candidates explaining that nothing changes for them semantically but they do become a more likely occurrence for Trickle ICE.

- o Limit the number of pairs to 100 to comply with 5245.

- o Added clarifications on the non-importance of how newly discovered candidates are trickled/sent to the remote party or if this is done at all.

- o Added transport expectations for trickled candidates as per Dale Worley’s recommendation.

C.4. Changes from draft-ivov-00

- o Specified that end-of-candidates is a media level attribute which can of course appear as session level, which is equivalent to having it appear in all m-lines. Also made end-of-candidates optional for cases such as aggressive nomination for controlled agents.

- o Added an example for ICE lite and Trickle ICE to illustrate how, when talking to an ICE lite agent doesn’t need to send or even discover any candidates.

- o Added an example for ICE lite and Trickle ICE to illustrate how, when talking to an ICE lite agent doesn’t need to send or even discover any candidates.

- o Added wording that explicitly states ICE lite agents have to be prepared to receive no candidates over signalling and that they should not freak out if this happens. (Closed the corresponding open issue).

- o It is now mandatory to use MID when trickling candidates and using m-line indexes is no longer allowed.

- o Replaced use of 0.0.0.0 to IP6 :: in order to avoid potential issues with RFC2543 SDP libraries that interpret 0.0.0.0 as an on-hold operation. Also changed the port number here from 1 to 9 since it already has a more appropriate meaning. (Port change suggested by Jonathan Lennox).

- o Closed the Open Issue about use about what to do with cands received after end-of-cands. Solution: ignore, do an ICE restart if you want to add something.

- o Added more terminology, including trickling, trickled candidates, half trickle, full trickle,

- o Added a reference to the SIP usage for Trickle ICE as requested at the Boston interim.

C.5. Changes from draft-rescorla-01

- o Brought back explicit use of Offer/Answer. There are no more attempts to try to do this in an O/A independent way. Also removed the use of ICE Descriptions.

- o Added SDP specification for trickled candidates, the trickle option and 0.0.0.0 addresses in m-lines, and end-of-candidates.

- o Support and Discovery. Changed that section to be less abstract. As discussed in IETF85, the draft now says implementations and usages need to either determine support in advance and directly use trickle, or do half trickle. Removed suggestion about use of discovery in SIP or about letting implementing protocols do what they want.

- o Defined Half Trickle. Added a section that says how it works. Mentioned that it only needs to happen in the first o/a (not necessary in updates), and added Jonathan’s comment about how it could, in some cases, offer more than half the improvement if you

can pre-gather part or all of your candidates before the user actually presses the call button.

- o Added a short section about subsequent offer/answer exchanges.
- o Added a short section about interactions with ICE Lite implementations.
- o Added two new entries to the open issues section.

C.6. Changes from draft-rescorla-00

- o Relaxed requirements about verifying support following a discussion on MMUSIC.
- o Introduced ICE descriptions in order to remove ambiguous use of 3264 language and inappropriate references to offers and answers.
- o Removed inappropriate assumption of adoption by RTCWEB pointed out by Martin Thomson.

Authors' Addresses

Emil Ivov
Jitsi
Strasbourg 67000
France

Phone: +33 6 72 81 15 55
Email: emcho@jitsi.org

Eric Rescorla
RTFM, Inc.
2064 Edgewood Drive
Palo Alto, CA 94303
USA

Phone: +1 650 678 2350
Email: ekr@rtfm.com

Justin Uberti
Google
747 6th St S
Kirkland, WA 98033
USA

Phone: +1 857 288 8888
Email: justin@uberti.name

Peter Saint-Andre
&yet

Email: peter@andyet.com
URI: https://andyet.com/

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 17, 2018

E. Ivov
Atlassian
E. Rescorla
RTFM, Inc.
J. Uberti
Google
P. Saint-Andre
Mozilla

April 15, 2018

Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol
draft-ietf-ice-trickle-21

Abstract

This document describes "Trickle ICE", an extension to the Interactive Connectivity Establishment (ICE) protocol that enables ICE agents to begin connectivity checks while they are still gathering candidates, by incrementally exchanging candidates over time instead of all at once. This method can considerably accelerate the process of establishing a communication session.

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 October 17, 2018.

Copyright Notice

Copyright (c) 2018 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.

Table of Contents

1. Introduction 3
2. Terminology 5
3. Determining Support for Trickle ICE 6
4. Generating the Initial ICE Description 7
5. Handling the Initial ICE Description and Generating the Initial ICE Response 7
6. Handling the Initial ICE Response 8
7. Forming Check Lists 8
8. Performing Connectivity Checks 8
9. Gathering and Conveying Newly Gathered Local Candidates . . . 9
10. Pairing Newly Gathered Local Candidates 10
11. Receiving Trickled Candidates 11
12. Inserting Trickled Candidate Pairs into a Check List 12
13. Generating an End-of-Candidates Indication 16
14. Receiving an End-of-Candidates Indication 17
15. Subsequent Exchanges and ICE Restarts 18
16. Half Trickle 18
17. Preserving Candidate Order while Trickling 19
18. Requirements for Using Protocols 20
19. IANA Considerations 21
20. Security Considerations 21
21. Acknowledgements 21
22. References 22
22.1. Normative References 22
22.2. Informative References 22
Appendix A. Interaction with Regular ICE 23
Appendix B. Interaction with ICE Lite 25
Appendix C. Changes from Earlier Versions 26
C.1. Changes from draft-ietf-ice-trickle-20 26
C.2. Changes from draft-ietf-ice-trickle-19 26
C.3. Changes from draft-ietf-ice-trickle-18 26
C.4. Changes from draft-ietf-ice-trickle-17 27
C.5. Changes from draft-ietf-ice-trickle-16 27
C.6. Changes from draft-ietf-ice-trickle-15 27
C.7. Changes from draft-ietf-ice-trickle-14 27
C.8. Changes from draft-ietf-ice-trickle-13 27
C.9. Changes from draft-ietf-ice-trickle-12 27
C.10. Changes from draft-ietf-ice-trickle-11 28

C.11. Changes from draft-ietf-ice-trickle-10 28
C.12. Changes from draft-ietf-ice-trickle-09 28
C.13. Changes from draft-ietf-ice-trickle-08 28
C.14. Changes from draft-ietf-ice-trickle-07 28
C.15. Changes from draft-ietf-ice-trickle-06 28
C.16. Changes from draft-ietf-ice-trickle-05 28
C.17. Changes from draft-ietf-ice-trickle-04 29
C.18. Changes from draft-ietf-ice-trickle-03 29
C.19. Changes from draft-ietf-ice-trickle-02 29
C.20. Changes from draft-ietf-ice-trickle-01 29
C.21. Changes from draft-ietf-ice-trickle-00 29
C.22. Changes from draft-mmusic-trickle-ice-02 29
C.23. Changes from draft-ivov-01 and draft-mmusic-00 30
C.24. Changes from draft-ivov-00 30
C.25. Changes from draft-rescorla-01 31
C.26. Changes from draft-rescorla-00 32
Authors' Addresses 32

1. Introduction

The Interactive Connectivity Establishment (ICE) protocol [rfc5245bis] describes how an ICE agent gathers candidates, exchanges candidates with a peer ICE agent, and creates candidate pairs. Once the pairs have been gathered, the ICE agent will perform connectivity checks, and eventually nominate and select pairs that will be used for sending and receiving data within a communication session.

Following the procedures in [rfc5245bis] can lead to somewhat lengthy establishment times for communication sessions, because candidate gathering often involves querying STUN servers [RFC5389] and allocating relayed candidates using TURN servers [RFC5766]. Although many ICE procedures can be completed in parallel, the pacing requirements from [rfc5245bis] still need to be followed.

This document defines "Trickle ICE", a supplementary mode of ICE operation in which candidates can be exchanged incrementally as soon as they become available (and simultaneously with the gathering of other candidates). Connectivity checks can also start as soon as candidate pairs have been created. Because Trickle ICE enables candidate gathering and connectivity checks to be done in parallel, the method can considerably accelerate the process of establishing a communication session.

This document also defines how to discover support for Trickle ICE, how the procedures in [rfc5245bis] are modified or supplemented when using Trickle ICE, and how a Trickle ICE agent can interoperate with an ICE agent compliant to [rfc5245bis].

This document does not define any protocol-specific usage of Trickle ICE. Instead, protocol-specific details for Trickle ICE are defined in separate usage documents. Examples of such documents are [I-D.ietf-mmusic-trickle-ice-sip] (which defines usage with the Session Initiation Protocol (SIP) [RFC3261] and the Session Description Protocol [RFC3261]) and [XEP-0176] (which defines usage with XMPP [RFC6120]). However, some of the examples in the document use SDP and the offer/answer model [RFC3264] to explain the underlying concepts.

The following diagram illustrates a successful Trickle ICE exchange with a using protocol that follows the offer/answer model:

```
        Alice                                             Bob
          |                     Offer                     |
          |-------------------------------------------->|
          |            Additional Candidates              |
          |-------------------------------------------->|
          |                     Answer                    |
          |<--------------------------------------------|
          |            Additional Candidates              |
          |<--------------------------------------------|
          | Additional Candidates and Connectivity Checks |
          |<------------------------------------------->|
          |<========== CONNECTION ESTABLISHED ===========>|
```

Figure 1: Flow

The main body of this document is structured to describe the behavior of Trickle ICE agents in roughly the order of operations and interactions during an ICE session:

1. Determining support for trickle ICE

2. Generating the initial ICE description

3. Handling the initial ICE description and generating the initial ICE response

4. Handling the initial ICE response

5. Forming check lists, pruning candidates, performing connectivity checks, etc.

6. Gathering and conveying candidates after the initial ICE description and response

7. Handling inbound trickled candidates

8. Generating and handling the end-of-candidates indication

9. Handling ICE restarts

There is quite a bit of operational experience with the technique behind Trickle ICE, going back as far as 2005 (when the XMPP Jingle extension defined a "dribble mode" as specified in [XEP-0176]); this document incorporates feedback from those who have implemented and deployed the technique over the years.

2. Terminology

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

This specification makes use of all terminology defined for Interactive Connectivity Establishment in [rfc5245bis]. In addition, it defines the following terms:

Full Trickle: The typical mode of operation for Trickle ICE agents, in which the initial ICE description can include any number of candidates (even zero candidates) and does not need to include a full generation of candidates as in half trickle.

Generation: All of the candidates conveyed within an ICE session.

Half Trickle: A Trickle ICE mode of operation in which the initiator gathers a full generation of candidates strictly before creating and conveying the initial ICE description. Once conveyed, this candidate information can be processed by regular ICE agents, which do not require support for Trickle ICE. It also allows Trickle ICE capable responders to still gather candidates and perform connectivity checks in a non-blocking way, thus providing roughly "half" the advantages of Trickle ICE. The half trickle mechanism is mostly meant for use when the responder’s support for Trickle ICE cannot be confirmed prior to conveying the initial ICE description.

ICE Description: Any attributes related to the ICE session (not candidates) required to configure an ICE agent. These include but are not limited to the username fragment, password, and other attributes.

Trickled Candidates: Candidates that a Trickle ICE agent conveys after conveying the initial ICE description or responding to the initial ICE description, but within the same ICE session. Trickled candidates can be conveyed in parallel with candidate gathering and connectivity checks.

Trickling: The act of incrementally conveying trickled candidates.

Empty Check List: A check list that initially does not contain any candidate pairs because they will be incrementally added as they are trickled. (This scenario does not arise with a regular ICE agent, because all candidate pairs are known when the agent creates the check list set).

3. Determining Support for Trickle ICE

To fully support Trickle ICE, using protocols SHOULD incorporate one of the following mechanisms so that implementations can determine whether Trickle ICE is supported:

1. Provide a capabilities discovery method so that agents can verify support of Trickle ICE prior to initiating a session (XMPP's Service Discovery [XEP-0030] is one such mechanism).

2. Make support for Trickle ICE mandatory so that user agents can assume support.

If a using protocol does not provide a method of determining ahead of time whether Trickle ICE is supported, agents can make use of the half trickle procedure described in Section 16.

Prior to conveying the initial ICE description, agents that implement using protocols that support capabilities discovery can attempt to verify whether or not the remote party supports Trickle ICE. If an agent determines that the remote party does not support Trickle ICE, it MUST fall back to using regular ICE or abandon the entire session.

Even if a using protocol does not include a capabilities discovery method, a user agent can provide an indication within the ICE description that it supports Trickle ICE by communicating an ICE option of 'trickle'. This token MUST be provided either at the session level or, if at the data stream level, for every data stream (an agent MUST NOT specify Trickle ICE support for some data streams but not others). Note: The encoding of the 'trickle' ICE option, and the message(s) used to carry it to the peer, are protocol specific; for instance, the encoding for the Session Description Protocol (SDP) [RFC4566] is defined in [I-D.ietf-mmusic-trickle-ice-sip].

Dedicated discovery semantics and half trickle are needed only prior to initiation of an ICE session. After an ICE session is established and Trickle ICE support is confirmed for both parties, either agent can use full trickle for subsequent exchanges (see also Section 15).

4. Generating the Initial ICE Description

An ICE agent can start gathering candidates as soon as it has an indication that communication is imminent (e.g., a user interface cue or an explicit request to initiate a communication session). Unlike in regular ICE, in Trickle ICE implementations do not need to gather candidates in a blocking manner. Therefore, unless half trickle is being used, the user experience is improved if the initiating agent generates and transmits its initial ICE description as early as possible (thus enabling the remote party to start gathering and trickling candidates).

An initiator MAY include any mix of candidates when conveying the initial ICE description. This includes the possibility of conveying all the candidates the initiator plans to use (as in half trickle), conveying only a publicly-reachable IP address (e.g., a candidate at a data relay that is known to not be behind a firewall), or conveying no candidates at all (in which case the initiator can obtain the responder’s initial candidate list sooner and the responder can begin candidate gathering more quickly).

For candidates included in the initial ICE description, the methods for calculating priorities and foundations, determining redundancy of candidates, and the like work just as in regular ICE [rfc5245bis].

5. Handling the Initial ICE Description and Generating the Initial ICE Response

When a responder receives the initial ICE description, it will first check if the ICE description or initiator indicates support for Trickle ICE as explained in Section 3. If not, the responder MUST process the initial ICE description according to regular ICE procedures [rfc5245bis] (or, if no ICE support is detected at all, according to relevant processing rules for the using protocol, such as offer/answer processing rules [RFC3264]). However, if support for Trickle ICE is confirmed, a responder will automatically assume support for regular ICE as well.

If the initial ICE description indicates support for Trickle ICE, the responder will determine its role and start gathering and prioritizing candidates; while doing so, it will also respond by conveying an initial ICE response, so that both the initiator and the responder can form check lists and begin connectivity checks.

A responder can respond to the initial ICE description at any point while gathering candidates. The initial ICE response MAY contain any set of candidates, including all candidates or no candidates. (The benefit of including no candidates is to convey the initial ICE response as quickly as possible, so that both parties can consider the ICE session to be under active negotiation as soon as possible.)

As noted in Section 3, in using protocols that use SDP the initial ICE response can indicate support for Trickle ICE by including a token of "trickle" in the ice-options attribute.

6. Handling the Initial ICE Response

When processing the initial ICE response, the initiator follows regular ICE procedures to determine its role, after which it forms check lists (Section 7) and performs connectivity checks (Section 8).

7. Forming Check Lists

According to regular ICE procedures [rfc5245bis], in order for candidate pairing to be possible and for redundant candidates to be pruned, the candidates would need to be provided in the initial ICE description and initial ICE response. By contrast, under Trickle ICE check lists can be empty until candidates are conveyed or received. Therefore a Trickle ICE agent handles check list formation and candidate pairing in a slightly different way than a regular ICE agent: the agent still forms the check lists, but it populates a given check list only after it actually has candidate pairs for that check list. Every check list is initially placed in the Running state, even if the check list is empty (this is consistent with Section 6.1.2.1 of [rfc5245bis]).

8. Performing Connectivity Checks

As specified in [rfc5245bis], whenever timer Ta fires, only check lists in the Running state will be picked when scheduling connectivity checks for candidate pairs. Therefore, a Trickle ICE agent MUST keep each check list in the Running state as long as it expects candidate pairs to be incrementally added to the check list. After that, the check list state is set according to the procedures in [rfc5245bis].

Whenever timer Ta fires and an empty check list is picked, no action is performed for the list. Without waiting for timer Ta to expire again, the agent selects the next check list in the Running state, in accordance with Section 6.1.4.2 of [rfc5245bis].

Section 7.2.5.3.3 of [rfc5245bis] requires that agents update check lists and timer states upon completing a connectivity check transaction. During such an update, regular ICE agents would set the state of a check list to Failed if both of the following two conditions are satisfied:

- o all of the pairs in the check list are either in the Failed state or Succeeded state; and

- o there is not a pair in the valid list for each component of the data stream.

With Trickle ICE, the above situation would often occur when candidate gathering and trickling are still in progress, even though it is quite possible that future checks will succeed. For this reason, Trickle ICE agents add the following conditions to the above list:

- o all candidate gathering has completed and the agent is not expecting to discover any new local candidates; and

- o the remote agent has conveyed an end-of-candidates indication for that check list as described in Section 13.

9. Gathering and Conveying Newly Gathered Local Candidates

After Trickle ICE agents have conveyed initial ICE descriptions and initial ICE responses, they will most likely continue gathering new local candidates as STUN, TURN, and other non-host candidate gathering mechanisms begin to yield results. Whenever an agent discovers such a new candidate it will compute its priority, type, foundation, and component ID according to regular ICE procedures.

The new candidate is then checked for redundancy against the existing list of local candidates. If its transport address and base match those of an existing candidate, it will be considered redundant and will be ignored. This would often happen for server reflexive candidates that match the host addresses they were obtained from (e.g., when the latter are public IPv4 addresses). Contrary to regular ICE, Trickle ICE agents will consider the new candidate redundant regardless of its priority.

Next the agent "trickles" the newly discovered candidate(s) to the remote agent. The actual delivery of the new candidates is handled by a using protocol such as SIP or XMPP. Trickle ICE imposes no restrictions on the way this is done (e.g., some using protocols might choose not to trickle updates for server reflexive candidates and instead rely on the discovery of peer reflexive ones).

When candidates are trickled, the using protocol MUST deliver each candidate (and any end-of-candidates indication as described in Section 13) to the receiving Trickle ICE implementation exactly once and in the same order it was conveyed. If the using protocol provides any candidate retransmissions, they need to be hidden from the ICE implementation.

Also, candidate trickling needs to be correlated to a specific ICE session, so that if there is an ICE restart, any delayed updates for a previous session can be recognized as such and ignored by the receiving party. For example, using protocols that signal candidates via SDP might include a Username Fragment value in the corresponding a=candidate line, such as:

```
a=candidate:1 1 UDP 2130706431 2001:db8::1 5000 typ host ufrag 8hhY
```

Or, as another example, WebRTC implementations might include a Username Fragment in the JavaScript objects that represent candidates.

Note: The using protocol needs to provide a mechanism for both parties to indicate and agree on the ICE session in force (as identified by the Username Fragment and Password combination) so that they have a consistent view of which candidates are to be paired. This is especially important in the case of ICE restarts (see Section 15).

Note: A using protocol might prefer not to trickle server reflexive candidates to entities that are known to be publicly accessible and where sending a direct STUN binding request is likely to reach the destination faster than the trickle update that travels through the signaling path.

10. Pairing Newly Gathered Local Candidates

As a Trickle ICE agent gathers local candidates, it needs to form candidate pairs; this works as described in the ICE specification [rfc5245bis], with the following provisos:

1. A Trickle ICE agent MUST NOT pair a local candidate until it has been trickled to the remote party.

2. Once the agent has conveyed the local candidate to the remote party, the agent checks if any remote candidates are currently known for this same stream and component. If not, the agent

merely adds the new candidate to the list of local candidates (without pairing it).

3. Otherwise, if the agent has already learned of one or more remote candidates for this stream and component, it attempts to pair the new local candidate as described in the ICE specification [rfc5245bis].

4. If a newly formed pair has a local candidate whose type is server reflexive, the agent MUST replace the local candidate with its base before completing the relevant redundancy tests.

5. The agent prunes redundant pairs by following the rules in Section 6.1.2.4 of [rfc5245bis], but checks existing pairs only if they have a state of Waiting or Frozen; this avoids removal of pairs for which connectivity checks are in flight (a state of In-Progress) or for which connectivity checks have already yielded a definitive result (a state of Succeeded or Failed).

6. If after the relevant redundancy tests the check list where the pair is to be added already contains the maximum number of candidate pairs (100 by default as per [rfc5245bis]), the agent SHOULD discard any pairs in the Failed state to make room for the new pair. If there are no such pairs, the agent SHOULD discard a pair with a lower priority than the new pair in order to make room for the new pair, until the number of pairs is equal to the maximum number of pairs. This processing is consistent with Section 6.1.2.5 of [rfc5245bis].

11. Receiving Trickled Candidates

At any time during an ICE session, a Trickle ICE agent might receive new candidates from the remote agent, from which it will attempt to form a candidate pair; this works as described in the ICE specification [rfc5245bis], with the following provisos:

1. The agent checks if any local candidates are currently known for this same stream and component. If not, the agent merely adds the new candidate to the list of remote candidates (without pairing it).

2. Otherwise, if the agent has already gathered one or more local candidates for this stream and component, it attempts to pair the new remote candidate as described in the ICE specification [rfc5245bis].

3. If a newly formed pair has a local candidate whose type is server reflexive, the agent MUST replace the local candidate with its base before completing the redundancy check in the next step.

4. The agent prunes redundant pairs as described below, but checks existing pairs only if they have a state of Waiting or Frozen; this avoids removal of pairs for which connectivity checks are in flight (a state of In-Progress) or for which connectivity checks have already yielded a definitive result (a state of Succeeded or Failed).

   A. If the agent finds a redundancy between two pairs and one of those pairs contains a newly received remote candidate whose type is peer reflexive, the agent SHOULD discard the pair containing that candidate, set the priority of the existing pair to the priority of the discarded pair, and re-sort the check list. (This policy helps to eliminate problems with remote peer reflexive candidates for which a STUN binding request is received before signaling of the candidate is trickled to the receiving agent, such as a different view of pair priorities between the local agent and the remote agent, since the same candidate could be perceived as peer reflexive by one agent and as server reflexive by the other agent.)

   B. The agent then applies the rules defined in Section 6.1.2.4 of [rfc5245bis].

5. If after the relevant redundancy tests the check list where the pair is to be added already contains the maximum number of candidate pairs (100 by default as per [rfc5245bis]), the agent SHOULD discard any pairs in the Failed state to make room for the new pair. If there are no such pairs, the agent SHOULD discard a pair with a lower priority than the new pair in order to make room for the new pair, until the number of pairs is equal to the maximum number of pairs. This processing is consistent with Section 6.1.2.5 of [rfc5245bis].

12. Inserting Trickled Candidate Pairs into a Check List

After a local agent has trickled a candidate and formed a candidate pair from that local candidate (Section 9), or after a remote agent has received a trickled candidate and formed a candidate pair from that remote candidate (Section 11), a Trickle ICE agent adds the new candidate pair to a check list as defined in this section.

As an aid to understanding the procedures defined in this section, consider the following tabular representation of all check lists in

an agent (note that initially for one of the foundations, i.e., f5, there are no candidate pairs):

	f1	f2	f3	f4	f5
s1 (Audio.RTP)	F	F	F		
s2 (Audio.RTCP)	F	F	F	F	
s3 (Video.RTP)	F				
s4 (Video.RTCP)	F				

Figure 2: Example of Check List State

Each row in the table represents a component for a given data stream (e.g., s1 and s2 might be the RTP and RTCP components for audio) and thus a single check list in the check list set. Each column represents one foundation. Each cell represents one candidate pair. In the tables shown in this section, "F" stands for "frozen", "W" stands for "waiting", and "S" stands for "succeeded"; in addition, "^^" is used to notate newly-added candidate pairs.

When an agent commences ICE processing, in accordance with Section 6.1.2.6 of [rfc5245bis], for each foundation it will unfreeze the pair with the lowest component ID and, if the component IDs are equal, with the highest priority (this is the topmost candidate pair in every column). This initial state is shown in the following table.

	f1	f2	f3	f4	f5
s1 (Audio.RTP)	W	W	W		
s2 (Audio.RTCP)	F	F	F	W	
s3 (Video.RTP)	F				
s4 (Video.RTCP)	F				

Figure 3: Initial Check List State

Then, as the checks proceed (see Section 7.2.5.4 of [rfc5245bis]), for each pair that enters the Succeeded state (denoted here by "S"), the agent will unfreeze all pairs for all data streams with the same foundation (e.g., if the pair in column 1, row 1 succeeds then the agent will unfreeze the pair in column 1, rows 2, 3, and 4).

	f1	f2	f3	f4	f5
s1 (Audio.RTP)	S	W	W		
s2 (Audio.RTCP)	W	F	F	W	
s3 (Video.RTP)	W				
s4 (Video.RTCP)	W				

Figure 4: Check List State with Succeeded Candidate Pair

Trickle ICE preserves all of these rules as they apply to "static" check list sets. This implies that if a Trickle ICE agent were to begin connectivity checks with all of its pairs already present, the way that pair states change is indistinguishable from that of a regular ICE agent.

Of course, the major difference with Trickle ICE is that check list sets can be dynamically updated because candidates can arrive after connectivity checks have started. When this happens, an agent sets the state of the newly formed pair as described below.

Rule 1: If the newly formed pair has the lowest component ID and, if the component IDs are equal, the highest priority of any candidate pair for this foundation (i.e., if it is the topmost pair in the column), set the state to Waiting. For example, this would be the case if the newly formed pair were placed in column 5, row 1. This rule is consistent with Section 6.1.2.6 of [rfc5245bis].

	f1	f2	f3	f4	f5
s1 (Audio.RTP)	S	W	W		^W^
s2 (Audio.RTCP)	W	F	F	W	
s3 (Video.RTP)	W				
s4 (Video.RTCP)	W				

Figure 5: Check List State with Newly Formed Pair, Rule 1

Rule 2: If there is at least one pair in the Succeeded state for this foundation, set the state to Waiting. For example, this would be the case if the pair in column 5, row 1 succeeded and the newly formed pair were placed in column 5, row 2. This rule is consistent with Section 7.2.5.3.3 of [rfc5245bis].

	f1	f2	f3	f4	f5
s1 (Audio.RTP)	S	W	W		S
s2 (Audio.RTCP)	W	F	F	W	^W^
s3 (Video.RTP)	W				
s4 (Video.RTCP)	W				

Figure 6: Check List State with Newly Formed Pair, Rule 2

Rule 3: In all other cases, set the state to Frozen. For example, this would be the case if the newly formed pair were placed in column 3, row 3.

```
+-----------------+------+------+------+------+------+
|                 |  f1  |  f2  |  f3  |  f4  |  f5  |
+-----------------+------+------+------+------+------+
| s1 (Audio.RTP)  |  S   |  W   |  W   |      |  S   |
+-----------------+------+------+------+------+------+
| s2 (Audio.RTCP) |  W   |  F   |  F   |  W   |  W   |
+-----------------+------+------+------+------+------+
| s3 (Video.RTP)  |  W   |      | ^F^  |      |      |
+-----------------+------+------+------+------+------+
| s4 (Video.RTCP) |  W   |      |      |      |      |
+-----------------+------+------+------+------+------+
```

Figure 7: Check List State with Newly Formed Pair, Rule 3

13. Generating an End-of-Candidates Indication

Once all candidate gathering is completed or expires for an ICE session associated with a specific data stream, the agent will generate an "end-of-candidates" indication for that session and convey it to the remote agent via the signaling channel. Although the exact form of the indication depends on the using protocol, the indication MUST specify the generation (Username Fragment and Password combination) so that an agent can correlate the end-of-candidates indication with a particular ICE session. The indication can be conveyed in the following ways:

- o As part of an initiation request (which would typically be the case with the initial ICE description for half trickle)

- o Along with the last candidate an agent can send for a stream

- o As a standalone notification (e.g., after STUN Binding requests or TURN Allocate requests to a server time out and the agent is no longer actively gathering candidates)

Conveying an end-of-candidates indication in a timely manner is important in order to avoid ambiguities and speed up the conclusion of ICE processing. In particular:

- o A controlled Trickle ICE agent SHOULD convey an end-of-candidates indication after it has completed gathering for a data stream, unless ICE processing terminates before the agent has had a chance to complete gathering.

- o A controlling agent MAY conclude ICE processing prior to conveying end-of-candidates indications for all streams. However, it is RECOMMENDED for a controlling agent to convey end-of-candidates

indications whenever possible for the sake of consistency and to keep middleboxes and controlled agents up-to-date on the state of ICE processing.

When conveying an end-of-candidates indication during trickling (rather than as a part of the initial ICE description or a response thereto), it is the responsibility of the using protocol to define methods for associating the indication with one or more specific data streams.

An agent MAY also choose to generate an end-of-candidates indication before candidate gathering has actually completed, if the agent determines that gathering has continued for more than an acceptable period of time. However, an agent MUST NOT convey any more candidates after it has conveyed an end-of-candidates indication.

When performing half trickle, an agent SHOULD convey an end-of-candidates indication together with its initial ICE description unless it is planning to potentially trickle additional candidates (e.g., in case the remote party turns out to support Trickle ICE).

After an agent conveys the end-of-candidates indication, it will update the state of the corresponding check list as explained in Section 8. Past that point, an agent MUST NOT trickle any new candidates within this ICE session. Therefore, adding new candidates to the negotiation is possible only through an ICE restart (see Section 15).

This specification does not override regular ICE semantics for concluding ICE processing. Therefore, even if end-of-candidates indications are conveyed, an agent will still need to go through pair nomination. Also, if pairs have been nominated for components and data streams, ICE processing MAY still conclude even if end-of-candidates indications have not been received for all streams. In all cases, an agent MUST NOT trickle any new candidates within an ICE session after nomination of a candidate pair as described in Section 8.1.1 of [rfc5245bis].

14. Receiving an End-of-Candidates Indication

Receiving an end-of-candidates indication enables an agent to update check list states and, in case valid pairs do not exist for every component in every data stream, determine that ICE processing has failed. It also enables an agent to speed up the conclusion of ICE processing when a candidate pair has been validated but it involves the use of lower-preference transports such as TURN. In such situations, an implementation MAY choose to wait and see if higher-priority candidates are received; in this case the end-of-candidates

indication provides a notification that such candidates are not forthcoming.

When an agent receives an end-of-candidates indication for a specific data stream, it will update the state of the relevant check list as per Section 8 (which might lead to some check lists being marked as Failed). If the check list is still in the Running state after the update, the agent will persist the fact that an end-of-candidates indication has been received and take it into account in future updates to the check list.

After an agent has received an end-of-candidates indication, it MUST ignore any newly received candidates for that data stream or data session.

15. Subsequent Exchanges and ICE Restarts

Before conveying an end-of-candidates indication, either agent MAY convey subsequent candidate information at any time allowed by the using protocol. When this happens, agents will use [rfc5245bis] semantics (e.g., checking of the Username Fragment and Password combination) to determine whether or not the new candidate information requires an ICE restart.

If an ICE restart occurs, the agents can assume that Trickle ICE is still supported if support was determined previously, and thus can engage in Trickle ICE behavior as they would in an initial exchange of ICE descriptions where support was determined through a capabilities discovery method.

16. Half Trickle

In half trickle, the initiator conveys the initial ICE description with a usable but not necessarily full generation of candidates. This ensures that the ICE description can be processed by a regular ICE responder and is mostly meant for use in cases where support for Trickle ICE cannot be confirmed prior to conveying the initial ICE description. The initial ICE description indicates support for Trickle ICE, so that the responder can respond with something less than a full generation of candidates and then trickle the rest. The initial ICE description for half trickle can contain an end-of-candidates indication, although this is not mandatory because if trickle support is confirmed then the initiator can choose to trickle additional candidates before it conveys an end-of-candidates indication.

The half trickle mechanism can be used in cases where there is no way for an agent to verify in advance whether a remote party supports

Trickle ICE. Because the initial ICE description contain a full generation of candidates, it can thus be handled by a regular ICE agent, while still allowing a Trickle ICE agent to use the optimization defined in this specification. This prevents negotiation from failing in the former case while still giving roughly half the Trickle ICE benefits in the latter.

Use of half trickle is only necessary during an initial exchange of ICE descriptions. After both parties have received an ICE description from their peer, they can each reliably determine Trickle ICE support and use it for all subsequent exchanges (see Section 15).

In some instances, using half trickle might bring more than just half the improvement in terms of user experience. This can happen when an agent starts gathering candidates upon user interface cues that the user will soon be initiating an interaction, such as activity on a keypad or the phone going off hook. This would mean that some or all of the candidate gathering could be completed before the agent actually needs to convey the candidate information. Because the responder will be able to trickle candidates, both agents will be able to start connectivity checks and complete ICE processing earlier than with regular ICE and potentially even as early as with full trickle.

However, such anticipation is not always possible. For example, a multipurpose user agent or a WebRTC web page where communication is a non-central feature (e.g., calling a support line in case of a problem with the main features) would not necessarily have a way of distinguishing between call intentions and other user activity. In such cases, using full trickle is most likely to result in an ideal user experience. Even so, using half trickle would be an improvement over regular ICE because it would result in a better experience for responders.

17. Preserving Candidate Order while Trickling

One important aspect of regular ICE is that connectivity checks for a specific foundation and component are attempted simultaneously by both agents, so that any firewalls or NATs fronting the agents would whitelist both endpoints and allow all except for the first ("suicide") packets to go through. This is also important to unfreezing candidates at the right time. While not crucial, preserving this behavior in Trickle ICE is likely to improve ICE performance.

To achieve this, when trickling candidates, agents SHOULD respect the order of components as reflected by their component IDs; that is, candidates for a given component SHOULD NOT be conveyed prior to

candidates for a component with a lower ID number within the same foundation. In addition, candidates SHOULD be paired, following the procedures in Section 12, in the same order they are conveyed.

For example, the following SDP description contains two components (RTP and RTCP) and two foundations (host and server reflexive):

```
v=0
o=jdoe 2890844526 2890842807 IN IP4 10.0.1.1
s=
c=IN IP4 10.0.1.1
t=0 0
a=ice-pwd:asd88fgpdd777uzjYhagZg
a=ice-ufrag:8hhY
m=audio 5000 RTP/AVP 0
a=rtpmap:0 PCMU/8000
a=candidate:1 1 UDP 2130706431 10.0.1.1 5000 typ host
a=candidate:1 2 UDP 2130706431 10.0.1.1 5001 typ host
a=candidate:2 1 UDP 1694498815 192.0.2.3 5000 typ srflx
    raddr 10.0.1.1 rport 8998
a=candidate:2 2 UDP 1694498815 192.0.2.3 5001 typ srflx
    raddr 10.0.1.1 rport 8998
```

For this candidate information the RTCP host candidate would not be conveyed prior to the RTP host candidate. Similarly the RTP server reflexive candidate would be conveyed together with or prior to the RTCP server reflexive candidate.

18. Requirements for Using Protocols

In order to fully enable the use of Trickle ICE, this specification defines the following requirements for using protocols.

- o A using protocol SHOULD provide a way for parties to advertise and discover support for Trickle ICE before an ICE session begins (see Section 3).

- o A using protocol MUST provide methods for incrementally conveying (i.e., "trickling") additional candidates after conveying the initial ICE description (see Section 9).

- o A using protocol MUST deliver each trickled candidate or end-of-candidates indication exactly once and in the same order it was conveyed (see Section 9).

- o A using protocol MUST provide a mechanism for both parties to indicate and agree on the ICE session in force (see Section 9).

- o A using protocol MUST provide a way for parties to communicate the end-of-candidates indication, which MUST specify the particular ICE session to which the indication applies (see Section 13).

19. IANA Considerations

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: trickle

Contact: IESG, iesg@ietf.org

Change control: IESG

Description: An ICE option of "trickle" indicates support for incremental communication of ICE candidates.

Reference: RFC XXXX

20. Security Considerations

This specification inherits most of its semantics from [rfc5245bis] and as a result all security considerations described there apply to Trickle ICE.

If the privacy implications of revealing host addresses on an endpoint device are a concern (see for example the discussion in [I-D.ietf-rtcweb-ip-handling] and in Section 19 of [rfc5245bis]), agents can generate ICE descriptions that contain no candidates and then only trickle candidates that do not reveal host addresses (e.g., relayed candidates).

21. Acknowledgements

The authors would like to thank Bernard Aboba, Flemming Andreasen, Rajmohan Banavi, Taylor Brandstetter, Philipp Hancke, Christer Holmberg, Ari Keranen, Paul Kyzivat, Jonathan Lennox, Enrico Marocco, Pal Martinsen, Nils Ohlmeier, Thomas Stach, Peter Thatcher, Martin Thomson, Brandon Williams, and Dale Worley for their reviews and suggestions on improving this document. Sarah Banks, Roni Even, and David Mandelberg completed opsdir, genart, and security reviews, respectively. Thanks also to Ari Keranen and Peter Thatcher in their

role as chairs, and Ben Campbell in his role as responsible Area Director.

22. References

22.1. Normative References

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

[rfc5245bis] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", draft-ietf-ice-rfc5245bis-20 (work in progress), March 2018.

22.2. Informative References

[I-D.ietf-mmusic-trickle-ice-sip] Ivov, E., Stach, T., Marocco, E., and C. Holmberg, "A Session Initiation Protocol (SIP) usage for Trickle ICE", draft-ietf-mmusic-trickle-ice-sip-14 (work in progress), February 2018.

[I-D.ietf-rtcweb-ip-handling] Uberti, J. and G. Shieh, "WebRTC IP Address Handling Requirements", draft-ietf-rtcweb-ip-handling-06 (work in progress), March 2018.

[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>.

[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>.

[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>.

[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>.

[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>.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <https://www.rfc-editor.org/info/rfc6120>.

[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>.

[XEP-0030] Hildebrand, J., Millard, P., Eatmon, R., and P. Saint-Andre, "XEP-0030: Service Discovery", XEP XEP-0030, June 2008.

[XEP-0176] Beda, J., Ludwig, S., Saint-Andre, P., Hildebrand, J., Egan, S., and R. McQueen, "XEP-0176: Jingle ICE-UDP Transport Method", XEP XEP-0176, June 2009.

Appendix A. Interaction with Regular ICE

The ICE protocol was designed to be flexible enough to work in and adapt to as many network environments as possible. Despite that flexibility, ICE as specified in [rfc5245bis] does not by itself support trickle ICE. This section describes how trickling of candidates interacts with ICE.

[rfc5245bis] describes the conditions required to update check lists and timer states while an ICE agent is in the Running state. These conditions are verified upon transaction completion and one of them stipulates that:

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

This could be a problem and cause ICE processing to fail prematurely in a number of scenarios. Consider the following case:

1. Alice and Bob are both located in different networks with Network Address Translation (NAT). Alice and Bob themselves have different address but both networks use the same private internet block (e.g., the "20-bit block" 172.16/12 specified in [RFC1918]).

2. Alice conveys to Bob the candidate 172.16.0.1 which also happens to correspond to an existing host on Bob’s network.

3. Bob creates a check list consisting solely of 172.16.0.1 and starts checks.

4. These checks reach the host at 172.16.0.1 in Bob’s network, which responds with an ICMP "port unreachable" error; per [rfc5245bis] Bob marks the transaction as Failed.

At this point the check list only contains Failed candidates and the valid list is empty. This causes the data stream and potentially all ICE processing to fail, even though if Trickle ICE agents could subsequently convey candidates that would cause previously empty check lists to become non-empty.

A similar race condition would occur if the initial ICE description from Alice contain only candidates that can be determined as unreachable from any of the candidates that Bob has gathered (e.g., this would be the case if Bob’s candidates only contain IPv4 addresses and the first candidate that he receives from Alice is an IPv6 one).

Another potential problem could arise when a non-trickle ICE implementation initiates an interaction with a Trickle ICE implementation. Consider the following case:

1. Alice’s client has a non-Trickle ICE implementation.

2. Bob’s client has support for Trickle ICE.

3. Alice and Bob are behind NATs with address-dependent filtering [RFC4787].

4. Bob has two STUN servers but one of them is currently unreachable.

After Bob's agent receives Alice's initial ICE description it would immediately start connectivity checks. It would also start gathering candidates, which would take a long time because of the unreachable STUN server. By the time Bob's answer is ready and conveyed to Alice, Bob's connectivity checks might have failed: until Alice gets Bob's answer, she won't be able to start connectivity checks and punch holes in her NAT. The NAT would hence be filtering Bob's checks as originating from an unknown endpoint.

Appendix B. Interaction with ICE Lite

The behavior of ICE lite agents that are capable of Trickle ICE does not require any particular rules other than those already defined in this specification and [rfc5245bis]. This section is hence provided only for informational purposes.

An ICE lite agent would generate candidate information as per [rfc5245bis] and would indicate support for Trickle ICE. Given that the candidate information will contain a full generation of candidates, it would also be accompanied by an end-of-candidates indication.

When performing full trickle, a full ICE implementation could convey the initial ICE description or response thereto with no candidates. After receiving a response that identifies the remote agent as an ICE lite implementation, the initiator can choose to not trickle any additional candidates. The same is also true in the case when the ICE lite agent initiates the interaction and the full ICE agent is the responder. In these cases the connectivity checks would be enough for the ICE lite implementation to discover all potentially useful candidates as peer reflexive. The following example illustrates one such ICE session using SDP syntax:

```
           ICE Lite                                          Bob
            Agent
              |   Offer (a=ice-lite a=ice-options:trickle)    |
              |---------------------------------------------->|
              |                                               |no cand
              |         Answer (a=ice-options:trickle)        |trickling
              |<----------------------------------------------|
              |              Connectivity Checks              |
              |<----------------------------------------------->|
     peer rflx|                                               |
    cand disco|                                               |
              |<========== CONNECTION ESTABLISHED ===========>|
```

Figure 8: Example

In addition to reducing signaling traffic this approach also removes the need to discover STUN bindings or make TURN allocations, which can considerably lighten ICE processing.

Appendix C. Changes from Earlier Versions

Note to the RFC Editor: please remove this section prior to publication as an RFC.

C.1. Changes from draft-ietf-ice-trickle-20

- o Slight corrections to hanlding of peer reflexive candidates.
- o Wordsmithing in a few sections.

C.2. Changes from draft-ietf-ice-trickle-19

- o Further clarified handling of remote peer reflexive candidates.
- o To improve readibility, renamed and restructured some sections and subsections, and modified some wording.

C.3. Changes from draft-ietf-ice-trickle-18

- o Cleaned up pairing and redundancy checking rules for newly discovered candidates per IESG feedback and WG discussion.
- o Improved wording in half trickle section.
- o Changed "not more than once" to "exactly once".

- o Changed NAT examples back to IPv4.

C.4. Changes from draft-ietf-ice-trickle-17

- o Simplified the rules for inserting a new pair in a check list.
- o Clarified it is not allowed to nominate a candidate pair after a pair has already been nominated (a.k.a. renomination or continuous nomination).
- o Removed some text that referenced older versions of rfc5245bis.
- o Removed some text that duplicated concepts and procedures specified in rfc5245bis.
- o Removed the ill-defined concept of stream order.
- o Shortened the introduction.

C.5. Changes from draft-ietf-ice-trickle-16

- o Made "ufrag" terminology consistent with 5245bis.
- o Applied in-order delivery rule to end-of-candidates indication.

C.6. Changes from draft-ietf-ice-trickle-15

- o Adjustments to address AD review feedback.

C.7. Changes from draft-ietf-ice-trickle-14

- o Minor modifications to track changes to ICE core.

C.8. Changes from draft-ietf-ice-trickle-13

- o Removed independent monitoring of check list "states" of frozen or active, since this is handled by placing a check list in the Running state defined in ICE core.

C.9. Changes from draft-ietf-ice-trickle-12

- o Specified that the end-of-candidates indication must include the generation (ufrag/pwd) to enable association with a particular ICE session.
- o Further editorial fixes to address WGLC feedback.

C.10. Changes from draft-ietf-ice-trickle-11

- o Editorial and terminological fixes to address WGLC feedback.

C.11. Changes from draft-ietf-ice-trickle-10

- o Minor editorial fixes.

C.12. Changes from draft-ietf-ice-trickle-09

- o Removed immediate unfreeze upon Fail.
- o Specified MUST NOT regarding ice-options.
- o Changed terminology regarding initial ICE parameters to avoid implementer confusion.

C.13. Changes from draft-ietf-ice-trickle-08

- o Reinstated text about in-order processing of messages as a requirement for signaling protocols.
- o Added IANA registration template for ICE option.
- o Corrected Case 3 rule in Section 8.1.1 to ensure consistency with regular ICE rules.
- o Added tabular representations to Section 8.1.1 in order to illustrate the new pair rules.

C.14. Changes from draft-ietf-ice-trickle-07

- o Changed "ICE description" to "candidate information" for consistency with 5245bis.

C.15. Changes from draft-ietf-ice-trickle-06

- o Addressed editorial feedback from chairs’ review.
- o Clarified terminology regarding generations.

C.16. Changes from draft-ietf-ice-trickle-05

- o Rewrote the text on inserting a new pair into a check list.

C.17. Changes from draft-ietf-ice-trickle-04

- Removed dependency on SDP and offer/answer model.
- Removed mentions of aggressive nomination, since it is deprecated in 5245bis.
- Added section on requirements for signaling protocols.
- Clarified terminology.
- Addressed various WG feedback.

C.18. Changes from draft-ietf-ice-trickle-03

- Provided more detailed description of unfreezing behavior, specifically how to replace pre-existing peer-reflexive candidates with higher-priority ones received via trickling.

C.19. Changes from draft-ietf-ice-trickle-02

- Adjusted unfreezing behavior when there are disparate foundations.

C.20. Changes from draft-ietf-ice-trickle-01

- Changed examples to use IPv6.

C.21. Changes from draft-ietf-ice-trickle-00

- Removed dependency on SDP (which is to be provided in a separate specification).
- Clarified text about the fact that a check list can be empty if no candidates have been sent or received yet.
- Clarified wording about check list states so as not to define new states for "Active" and "Frozen" because those states are not defined for check lists (only for candidate pairs) in ICE core.
- Removed open issues list because it was out of date.
- Completed a thorough copy edit.

C.22. Changes from draft-mmusic-trickle-ice-02

- Addressed feedback from Rajmohan Banavi and Brandon Williams.

- o Clarified text about determining support and about how to proceed if it can be determined that the answering agent does not support Trickle ICE.

- o Clarified text about check list and timer updates.

- o Clarified when it is appropriate to use half trickle or to send no candidates in an offer or answer.

- o Updated the list of open issues.

C.23. Changes from draft-ivov-01 and draft-mmusic-00

- o Added a requirement to trickle candidates by order of components to avoid deadlocks in the unfreezing algorithm.

- o Added an informative note on peer-reflexive candidates explaining that nothing changes for them semantically but they do become a more likely occurrence for Trickle ICE.

- o Limit the number of pairs to 100 to comply with 5245.

- o Added clarifications on the non-importance of how newly discovered candidates are trickled/sent to the remote party or if this is done at all.

- o Added transport expectations for trickled candidates as per Dale Worley’s recommendation.

C.24. Changes from draft-ivov-00

- o Specified that end-of-candidates is a media level attribute which can of course appear as session level, which is equivalent to having it appear in all m-lines. Also made end-of-candidates optional for cases such as aggressive nomination for controlled agents.

- o Added an example for ICE lite and Trickle ICE to illustrate how, when talking to an ICE lite agent doesn’t need to send or even discover any candidates.

- o Added an example for ICE lite and Trickle ICE to illustrate how, when talking to an ICE lite agent doesn’t need to send or even discover any candidates.

- o Added wording that explicitly states ICE lite agents have to be prepared to receive no candidates over signaling and that they

should not freak out if this happens. (Closed the corresponding open issue).

- o It is now mandatory to use MID when trickling candidates and using m-line indexes is no longer allowed.

- o Replaced use of 0.0.0.0 to IP6 :: in order to avoid potential issues with RFC2543 SDP libraries that interpret 0.0.0.0 as an on-hold operation. Also changed the port number here from 1 to 9 since it already has a more appropriate meaning. (Port change suggested by Jonathan Lennox).

- o Closed the Open Issue about use about what to do with cands received after end-of-cands. Solution: ignore, do an ICE restart if you want to add something.

- o Added more terminology, including trickling, trickled candidates, half trickle, full trickle,

- o Added a reference to the SIP usage for Trickle ICE as requested at the Boston interim.

C.25. Changes from draft-rescorla-01

- o Brought back explicit use of Offer/Answer. There are no more attempts to try to do this in an O/A independent way. Also removed the use of ICE Descriptions.

- o Added SDP specification for trickled candidates, the trickle option and 0.0.0.0 addresses in m-lines, and end-of-candidates.

- o Support and Discovery. Changed that section to be less abstract. As discussed in IETF85, the draft now says implementations and usages need to either determine support in advance and directly use trickle, or do half trickle. Removed suggestion about use of discovery in SIP or about letting implementing protocols do what they want.

- o Defined Half Trickle. Added a section that says how it works. Mentioned that it only needs to happen in the first o/a (not necessary in updates), and added Jonathan’s comment about how it could, in some cases, offer more than half the improvement if you can pre-gather part or all of your candidates before the user actually presses the call button.

- o Added a short section about subsequent offer/answer exchanges.

- o Added a short section about interactions with ICE Lite implementations.

- o Added two new entries to the open issues section.

C.26. Changes from draft-rescorla-00

- o Relaxed requirements about verifying support following a discussion on MMUSIC.

- o Introduced ICE descriptions in order to remove ambiguous use of 3264 language and inappropriate references to offers and answers.

- o Removed inappropriate assumption of adoption by RTCWEB pointed out by Martin Thomson.

Authors' Addresses

Emil Ivov
Atlassian
303 Colorado Street, #1600
Austin, TX 78701
USA

Phone: +1-512-640-3000
Email: eivov@atlassian.com

Eric Rescorla
RTFM, Inc.
2064 Edgewood Drive
Palo Alto, CA 94303
USA

Phone: +1 650 678 2350
Email: ekr@rtfm.com

Justin Uberti
Google
747 6th St S
Kirkland, WA 98033
USA

Phone: +1 857 288 8888
Email: justin@uberti.name

Peter Saint-Andre
Mozilla
P.O. Box 787
Parker, CO 80134
USA

Phone: +1 720 256 6756
Email: stpeter@mozilla.com
URI: https://www.mozilla.com/

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

P. Thatcher
H. Zhang
T. Brandstetter
Google

March 21, 2016

ICE Network Cost: Dynamically selecting ICE candidate pairs based on relative cost of network interfaces
draft-thatcher-ice-network-cost-00

Abstract

This document describes an extension to the Interactive Connectivity Establishment (ICE) that enables ICE agents to exchange information about the relative cost of network interfaces and dynamically choose the selected ICE candidate pair based on the cost of both the local and remote network interfaces. For example, if a cellular network interface has a higher cost than a Wi-Fi network interface, the ICE agents can use that information to prefer candidate pairs with Wi-Fi rather than cellular when possible, and only use cellular when necessary.

This document additionally describes a second piece of information, network ID, that goes along with the network cost and can be used to know when a network interface has changed, even if two network interfaces have the same network cost.

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 September 22, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
2. Terminology 3
3. Choosing a value for network cost and network ID 3
4. Signaling network cost and network ID 4
5. STUN attribute for network cost and network ID 4
6. Interpreting network cost and network ID 4
7. IANA Considerations 5
8. Security Considerations 5
9. Acknowledgements 5
10. Normative References 5
Authors' Addresses 5

1. Introduction

In certain network conditions, ICE agents may prefer to use a network interface with a lower cost (for a definition of cost chosen by the ICE agent, which need not be directly related to monetary costs). If the controlling side has such a preference, it can unilaterally nominate a candidate pair with the network interface with lower cost, but if either the controlling side has no such preference, or it would like to take the controlled side's preference into account, it cannot do so unless the controlled side provides information about its network cost.

Additionally, if the network interface of the controlled side changes (such as by using TURN mobility), the controlling side needs updated information from the controlled side.

The controlling side may also wish to select candidate pairs not only based on the relative cost between candidate pairs, but also the cost relative to the quality of the network path. For example, if Wi-Fi

has a much higher cost, but cellular is much higher quality, the controlling side may select cellular even though it’s higher cost. To do so, the controlled side must provide information about the network cost relative to the network quality. For example, if a network cost 10 is equivalent to 100ms network RTT, a Wi-Fi with cost 0 and RTT 150ms will have equal preference to a cellular with cost 10 and RTT 50ms.

Although the controlled side already communicates an ICE candidate priority, that candidate attribute doesn’t meet the needs of this situation for the following reasons:

- o Candidate priority affects ICE check ordering as well as candidate pair preference, which is undesirable in this situation, where the ICE check order should be maintained, but the candidate pair preference should be changed.

- o Candidate priority cannot change when the network interface changes (such as by using TURN mobility)

- o Candidate priority is only defined relative to other priorities, and can’t be compared against network quality in a meaningful way.

2. Terminology

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

This specification makes use of all terminology defined by the protocol for Interactive Connectivity Establishment in [RFC5245].

Network Cost A value indicating how much an ICE agent would prefer to not use a given network interface. This may be, but need not be related to monetary costs of using the network interface.

Network ID An ID that uniquely identifies a network interface.

3. Choosing a value for network cost and network ID

Network cost is an integer in the range 0-999, where larger values indicate a stronger preference for not using that network interface.

Each network interface SHOULD have a unique network ID, in the range of 0 to (2^16)-1.

4. Signaling network cost and network ID

ICE agents MUST signal network cost on each ICE candidate if the cost is non-zero. ICE agents MUST signal network ID on each ICE candidate.

For example, in an SDP candidate line, the attributes could be signaled as "network-cost 100 network-id 1".

5. STUN attribute for network cost and network ID

To communicate a change in network cost or to communicate network cost for peer reflexive candidates, the following STUN attribute is defined:

A 32-bit integer where the first 16 bits are the network ID and the second 16 bits are network cost:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Network ID           |         Network Cost          |
```

In the initial ICE checks, ICE agents MUST communicate a network cost and network ID if either is non-zero. The ICE agent MUST communicate new values in subsequent ICE checks if the network cost or network ID changes.

6. Interpreting network cost and network ID

If network cost is communicated via either signaling or STUN attribute, the controlling side SHOULD use the network cost of the controlled side as part of the criteria to determine which candidate pair to select. It SHOULD use network cost before using candidate priorities (network cost takes precedence over candidate priority), and it SHOULD NOT change the ICE check order based on network cost.

If the controlling side chooses to balance network cost against network quality, it is RECOMMENDED to treat a difference in network cost of 10 as equivalent of a change in network RTT of 100ms.

Any time the controlling side sees a change in the network cost from the controlled side, it MUST recalculate which candidate pair to select and nominate the newly selected candidate pair, if it has changed.

7. IANA Considerations

This specification requests no actions from IANA.

8. Security Considerations

TODO

9. Acknowledgements

TODO

10. Normative References

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

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

Authors' Addresses

Peter Thatcher
Google
747 6th St S
Kirkland, WA 98033
USA

Email: pthatcher@google.com

Honghai Zhang
Google
747 6th St S
Kirkland, WA 98033
USA

Email: honghaiz@google.com

Taylor Brandstetter
Google
747 6th St S
Kirkland, WA 98033
USA

Email: deadbeef@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 23, 2017

P. Thatcher
H. Zhang
T. Brandstetter
Google

September 19, 2016

ICE Network Cost: Dynamically selecting ICE candidate pairs based on relative cost of network interfaces
draft-thatcher-ice-network-cost-01

Abstract

This document describes an extension to the Interactive Connectivity Establishment (ICE) that enables ICE agents to exchange information about the relative cost of network interfaces and dynamically choose the selected ICE candidate pair based on the cost of both the local and remote network interfaces. For example, if a cellular network interface has a higher cost than a Wi-Fi network interface, the ICE agents can use that information to prefer candidate pairs with Wi-Fi rather than cellular when possible, and only use cellular when necessary.

This document additionally describes a second piece of information, network ID, that goes along with the network cost and can be used to know when a network interface has changed, even if two network interfaces have the same network cost.

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 March 23, 2017.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
2. Terminology 3
3. Choosing a value for network cost and network ID 3
4. Signaling network cost and network ID 4
5. STUN attribute for network cost and network ID 4
6. Interpreting network cost and network ID 4
7. IANA Considerations 5
8. Security Considerations 5
9. Acknowledgements 5
10. Normative References 5
Authors' Addresses 5

1. Introduction

In certain network conditions, ICE agents may prefer to use a network interface with a lower cost (for a definition of cost chosen by the ICE agent, which need not be directly related to monetary costs). If the controlling side has such a preference, it can unilaterally nominate a candidate pair with the network interface with lower cost, but if either the controlling side has no such preference, or it would like to take the controlled side's preference into account, it cannot do so unless the controlled side provides information about its network cost.

Additionally, if the network interface of the controlled side changes (such as by using TURN mobility), the controlling side needs updated information from the controlled side.

The controlling side may also wish to select candidate pairs not only based on the relative cost between candidate pairs, but also the cost relative to the quality of the network path. For example, if Wi-Fi

has a much higher cost, but cellular is much higher quality, the controlling side may select cellular even though it’s higher cost. To do so, the controlled side must provide information about the network cost relative to the network quality. For example, if a network cost 10 is equivalent to 100ms network RTT, a Wi-Fi with cost 0 and RTT 150ms will have equal preference to a cellular with cost 10 and RTT 50ms.

Although the controlled side already communicates an ICE candidate priority, that candidate attribute doesn’t meet the needs of this situation for the following reasons:

- o Candidate priority affects ICE check ordering as well as candidate pair preference, which is undesirable in this situation, where the ICE check order should be maintained, but the candidate pair preference should be changed.

- o Candidate priority cannot change when the network interface changes (such as by using TURN mobility)

- o Candidate priority is only defined relative to other priorities, and can’t be compared against network quality in a meaningful way.

2. Terminology

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

This specification makes use of all terminology defined by the protocol for Interactive Connectivity Establishment in [RFC5245].

Network Cost A value indicating how much an ICE agent would prefer to not use a given network interface. This may be, but need not be related to monetary costs of using the network interface.

Network ID An ID that uniquely identifies a network interface.

3. Choosing a value for network cost and network ID

Network cost is an integer in the range 0-999, where larger values indicate a stronger preference for not using that network interface.

Each network interface SHOULD have a unique network ID, in the range of 0 to (2^16)-1.

4. Signaling network cost and network ID

ICE agents MUST signal network cost on each ICE candidate if the cost is non-zero. ICE agents MUST signal network ID on each ICE candidate.

For example, in an SDP candidate line, the attributes could be signaled as "network-cost 100 network-id 1".

5. STUN attribute for network cost and network ID

To communicate a change in network cost or to communicate network cost for peer reflexive candidates, the following STUN attribute is defined:

A 32-bit integer where the first 16 bits are the network ID and the second 16 bits are network cost:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Network ID           |         Network Cost          |
```

In the initial ICE checks, ICE agents MUST communicate a network cost and network ID if either is non-zero. The ICE agent MUST communicate new values in subsequent ICE checks if the network cost or network ID changes.

6. Interpreting network cost and network ID

If network cost is communicated via either signaling or STUN attribute, the controlling side SHOULD use the network cost of the controlled side as part of the criteria to determine which candidate pair to select. It SHOULD use network cost before using candidate priorities (network cost takes precedence over candidate priority), and it SHOULD NOT change the ICE check order based on network cost.

If the controlling side chooses to balance network cost against network quality, it is RECOMMENDED to treat a difference in network cost of 10 as equivalent of a change in network RTT of 100ms.

Any time the controlling side sees a change in the network cost from the controlled side, it MUST recalculate which candidate pair to select and nominate the newly selected candidate pair, if it has changed.

7. IANA Considerations

This specification requests no actions from IANA.

8. Security Considerations

TODO

9. Acknowledgements

TODO

10. Normative References

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

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

Authors' Addresses

Peter Thatcher
Google
747 6th St S
Kirkland, WA 98033
USA

Email: pthatcher@google.com

Honghai Zhang
Google
747 6th St S
Kirkland, WA 98033
USA

Email: honghaiz@google.com

Taylor Brandstetter
Google
747 6th St S
Kirkland, WA 98033
USA

Email: deadbeef@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

P. Thatcher
H. Zhang
T. Brandstetter
Google

March 21, 2016

ICE Renomination: Dynamically selecting ICE candidate pairs
draft-thatcher-ice-renomination-00

Abstract

This document describes an extension to the Interactive Connectivity Establishment (ICE) that enables ICE agents to dynamically change the selected candidate pair of the controlled side by allowing the controlling side to nominate different candidate pairs over time as network conditions change.

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 September 22, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
2. Terminology 2
3. Renomination 2
4. "Nomination" attribute 3
5. IANA Considerations 3
6. Security Considerations 3
7. Acknowledgements 3
8. Normative References 3
Authors' Addresses 3

1. Introduction

ICE agents are either controlling or controlled. The controlling ICE agent can unilaterally select a given candidate pair at any time. But it cannot control what candidate pair the controlled ICE agent selects once the controlling ICE agent has nominated a candidate pair (with passive nomination) or nominated many candidate pairs (with aggressive nomination), with the exception that it may nominate a higher priority candidate pair with aggressive nomination. This greatly limits the controlling side's options.

For example, if an ICE agent selects and nominates a candidate pair over a cellular network, and then later connects to a Wi-Fi network and trickles ICE candidates for the Wi-Fi network, it may wish to select and nominate a candidate pair using Wi-Fi. If soon thereafter the Wi-Fi network disconnects and the ICE agent wishes to select and nominate the cellular candidate pair again, it would be unable to do with either passive or aggressive nomination.

2. Terminology

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

3. Renomination

We define a new ICE option called "renomination". When renomination is signaled, aggressive nomination is disabled, and the controlled side follows a rule of "last nomination wins". This allows the controlling side to send nominations for new candidate pairs at any time. The controlling side SHOULD send the new nomination until the STUN packet is acked to ensure that the renomination was received.

If one side signals "renomination" and the other does not understand it, then according to the rules of ICE, aggressive nomination is disabled and passive nomination is used, and the controlling side MUST NOT send more than one nomination.

4. "Nomination" attribute

To deal with out-of-order delivery of nominations, we define a new STUN attribute: "nomination" which includes a 24-bit integer in the 3 least significant bytes of the attribute.

The controlling side MAY include such an attribute when renominating. The controlled side MUST select the nomination with the largest value contained in the "nomination" attribute. Any value included takes precedence over the lack of a value.

5. IANA Considerations

This specification requests no actions from IANA.

6. Security Considerations

TODO

7. Acknowledgements

TODO

8. Normative References

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

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

Authors' Addresses

Peter Thatcher
Google
747 6th St S
Kirkland, WA 98033
USA

Email: pthatcher@google.com

Honghai Zhang
Google
747 6th St S
Kirkland, WA 98033
USA

Email: honghaiz@google.com

Taylor Brandstetter
Google
747 6th St S
Kirkland, WA 98033
USA

Email: deadbeef@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 23, 2017

P. Thatcher
H. Zhang
T. Brandstetter
Google

September 19, 2016

ICE Renomination: Dynamically selecting ICE candidate pairs
draft-thatcher-ice-renomination-01

Abstract

This document describes an extension to the Interactive Connectivity Establishment (ICE) that enables ICE agents to dynamically change the selected candidate pair of the controlled side by allowing the controlling side to nominate different candidate pairs over time as network conditions change.

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 March 23, 2017.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 2
2. Terminology 2
3. Renomination 2
4. "Nomination" attribute 3
5. IANA Considerations 3
6. Security Considerations 3
7. Acknowledgements 3
8. Normative References 3
Authors' Addresses 3

1. Introduction

ICE agents are either controlling or controlled. The controlling ICE agent can unilaterally select a given candidate pair at any time. But it cannot control what candidate pair the controlled ICE agent selects once the controlling ICE agent has nominated a candidate pair (with passive nomination) or nominated many candidate pairs (with aggressive nomination), with the exception that it may nominate a higher priority candidate pair with aggressive nomination. This greatly limits the controlling side's options.

For example, if an ICE agent selects and nominates a candidate pair over a cellular network, and then later connects to a Wi-Fi network and trickles ICE candidates for the Wi-Fi network, it may wish to select and nominate a candidate pair using Wi-Fi. If soon thereafter the Wi-Fi network disconnects and the ICE agent wishes to select and nominate the cellular candidate pair again, it would be unable to do with either passive or aggressive nomination.

2. Terminology

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

3. Renomination

We define a new ICE option called "renomination". When renomination is signaled, aggressive nomination is disabled, and the controlled side follows a rule of "last nomination wins". This allows the controlling side to send nominations for new candidate pairs at any time. The controlling side SHOULD send the new nomination until the STUN packet is acked to ensure that the renomination was received.

If one side signals "renomination" and the other does not understand it, then according to the rules of ICE, aggressive nomination is disabled and passive nomination is used, and the controlling side MUST NOT send more than one nomination.

4. "Nomination" attribute

To deal with out-of-order delivery of nominations, we define a new STUN attribute: "nomination" which includes a 24-bit integer in the 3 least significant bytes of the attribute.

The controlling side MAY include such an attribute when renominating. The controlled side MUST select the nomination with the largest value contained in the "nomination" attribute. Any value included takes precedence over the lack of a value.

5. IANA Considerations

This specification requests no actions from IANA.

6. Security Considerations

TODO

7. Acknowledgements

TODO

8. Normative References

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

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

Authors' Addresses

Peter Thatcher
Google
747 6th St S
Kirkland, WA 98033
USA

Email: pthatcher@google.com

Honghai Zhang
Google
747 6th St S
Kirkland, WA 98033
USA

Email: honghaiz@google.com

Taylor Brandstetter
Google
747 6th St S
Kirkland, WA 98033
USA

Email: deadbeef@google.com