[MS-ICE2]:

Interactive Connectivity Establishment (ICE) Extensions 2.0

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation ("this documentation") for protocols, file formats, data portability, computer languages, and standards support. Additionally, overview documents cover inter-protocol relationships and interactions.
- **Copyrights**. This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you can make copies of it in order to develop implementations of the technologies that are described in this documentation and can distribute portions of it in your implementations that use these technologies or in your documentation as necessary to properly document the implementation. You can also distribute in your implementation, with or without modification, any schemas, IDLs, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications documentation.
- **No Trade Secrets**. Microsoft does not claim any trade secret rights in this documentation.
- **Patents**. Microsoft has patents that might cover your implementations of the technologies described in the Open Specifications documentation. Neither this notice nor Microsoft's delivery of this documentation grants any licenses under those patents or any other Microsoft patents. However, a given Open Specifications document might be covered by the Microsoft Open Specifications Promise or the Microsoft Community Promise. If you would prefer a written license, or if the technologies described in this documentation are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **License Programs**. To see all of the protocols in scope under a specific license program and the associated patents, visit the Patent Map.
- **Trademarks**. The names of companies and products contained in this documentation might be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights. For a list of Microsoft trademarks, visit www.microsoft.com/trademarks.
- **Fictitious Names**. The example companies, organizations, products, domain names, email addresses, logos, people, places, and events that are depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than as specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications documentation does not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments, you are free to take advantage of them. Certain Open Specifications documents are intended for use in conjunction with publicly available standards specifications and network programming art and, as such, assume that the reader either is familiar with the aforementioned material or has immediate access to it.

Support. For questions and support, please contact dochelp@microsoft.com.

Revision Summary

Date	Revision History	Revision Class	Comments
12/12/2008	1.0	New	Initial version
2/13/2009	1.01	Minor	Revised and edited the technical content.
3/13/2009	1.02	Minor	Revised and edited the technical content.
7/13/2009	1.03	Major	Revised and edited the technical content
8/28/2009	1.04	Editorial	Revised and edited the technical content
11/6/2009	1.05	Editorial	Revised and edited the technical content
2/19/2010	1.06	Editorial	Revised and edited the technical content
3/31/2010	1.07	Major	Updated and revised the technical content
4/30/2010	2.08	Editorial	Revised and edited the technical content
6/7/2010	2.09	Editorial	Revised and edited the technical content
6/29/2010	2.10	Editorial	Changed language and formatting in the technical content.
7/23/2010	2.10	None	No changes to the meaning, language, or formatting of the technical content.
9/27/2010	3.0	Major	Significantly changed the technical content.
11/15/2010	3.0	None	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	3.0	None	No changes to the meaning, language, or formatting of the technical content.
3/18/2011	3.0	None	No changes to the meaning, language, or formatting of the technical content.
6/10/2011	3.0	None	No changes to the meaning, language, or formatting of the technical content.
1/20/2012	4.0	Major	Significantly changed the technical content.
4/11/2012	4.0	None	No changes to the meaning, language, or formatting of the technical content.
7/16/2012	4.0	None	No changes to the meaning, language, or formatting of the technical content.
10/8/2012	4.0.1	Editorial	Changed language and formatting in the technical content.
2/11/2013	4.0.1	None	No changes to the meaning, language, or formatting of the technical content.
7/30/2013	4.1	Minor	Clarified the meaning of the technical content.
11/18/2013	4.2	Minor	Clarified the meaning of the technical content.
2/10/2014	4.2	None	No changes to the meaning, language, or formatting of the technical content.
4/30/2014	4.2	None	No changes to the meaning, language, or formatting of the

Date	Revision History	Revision Class	Comments
			technical content.
7/31/2014	4.2	None	No changes to the meaning, language, or formatting of the technical content.
10/30/2014	4.2	None	No changes to the meaning, language, or formatting of the technical content.
3/30/2015	5.0	Major	Significantly changed the technical content.
9/4/2015	5.0	None	No changes to the meaning, language, or formatting of the technical content.
7/1/2016	6.0	Major	Significantly changed the technical content.
8/23/2016	6.0	None	No changes to the meaning, language, or formatting of the technical content.
9/14/2016	6.0	None	No changes to the meaning, language, or formatting of the technical content.
6/20/2017	7.0	Major	Significantly changed the technical content.
9/15/2017	8.0	Major	Significantly changed the technical content.

Table of Contents

1 Introduction

This document specifies the Interactive Connectivity Establishment (ICE) Extensions. This protocol consists of a set of proprietary extensions to the ICE protocol. ICE specifies a protocol for setting up **Real-Time Transport Protocol (RTP)** streams in a way that allows the streams to traverse **network address translation (NAT)** devices and firewalls.

Signaling protocols, such as **Session Initiation Protocol (SIP)**, are used to set up and negotiate media sessions. As part of setting up and negotiating the session, signaling protocols carry the IP addresses and ports of the call participants that receive RTP streams. Because NATs alter IP addresses and ports, the exchange of local IP addresses and ports might not be sufficient to establish connectivity. ICE uses protocols such as **Simple Traversal of UDP through NAT (STUN)** and **Traversal Using Relay NAT (TURN)** to establish and verify connectivity.

Sections 1.5, 1.8, 1.9, 2, and 3 of this specification are normative. All other sections and examples in this specification are informative.

1.1 Glossary

This document uses the following terms:

agent: A device that is connected to a computer network. Also referred to as an endpoint.

Aggressive Nomination: The process of selecting a valid **candidate pair** for media flow by sending **Simple Traversal of UDP through NAT (STUN)** binding requests that include the flag for every STUN binding request such that the first candidate pair that is validated is used for media flow.

answer: A message that is sent in response to an **offer** that is received from an offerer.

authentication: The act of proving an identity to a server while providing key material that binds the identity to subsequent communications.

base: The base of a host **candidate** is the host candidate itself. The base of server reflexive candidates and peer reflexive candidates is the host candidate from which they are derived. The base of a relayed candidate is the relayed candidate itself.

callee: An **endpoint** to which a call is initiated by a **caller**.

caller: An **endpoint** that initiates a call to establish a media session.

candidate: A set of **transport addresses** that form an atomic unit for use with a media session. For example, in the case of Real-Time Transport Protocol (RTP) there are two transport addresses for each candidate, one for RTP and another for the Real-Time Transport Control Protocol (RTCP). A candidate has properties such as type, priority, foundation, and **base**.

candidate pair: A set of candidates that is formed from a **local candidate** and a **remote candidate**.

Check List: An ordered list of **candidate pairs** that determines the order in which connectivity checks are performed for those candidate pairs.

component: A representation of a constituent **transport address** if a **candidate** consists of a set of transport addresses. For example, media streams that are based on the Real-Time Transfer Protocol (RTP) have two components, one for RTP and another for the Real-Time Transfer Control Protocol (RTCP).

connectivity check: A **Simple Traversal of UDP through NAT (STUN)** binding request that is sent to validate connectivity between the local and remote candidates in a **candidate pair**.

controlled agent: An Interactive Connectivity Establishment (ICE) agent that waits for the controlling agent to select the final **candidate pairs** to be used.

controlling agent: An Interactive Connectivity Establishment (ICE) agent that is responsible for selecting and signaling the final **candidate pair** that is selected by connectivity checks. The controlling agent signals the final **candidates** in a **Simple Traversal of UDP through NAT (STUN)** binding request and an updated offer. In a session, one of the agents is a controlling agent and the other agent is a controlled agent.

cyclic redundancy check (CRC): An algorithm used to produce a checksum (a small, fixed number of bits) against a block of data, such as a packet of network traffic or a block of a computer file. The CRC is a broad class of functions used to detect errors after transmission or storage. A CRC is designed to catch random errors, as opposed to intentional errors. If errors might be introduced by a motivated and intelligent adversary, a cryptographic hash function should be used instead.

default candidate: A **candidate** that is designated for streaming media before connectivity checks can be finished. The candidate that is most likely to stream media to the remote **endpoint** successfully is designated as the default candidate.

default candidate pair: A **candidate pair** that consists of the **caller's default candidate** and the **callee**'s default candidate.

endpoint: A device that is connected to a computer network.

final offer: An offer that is sent by a **caller** at the end of connectivity checks and carries the **local candidate** and the **remote candidate** that were selected for media flow.

foundation: A string that is a property associated with a **candidate**. The string is the same for candidates that are of the same type, protocol, and base IP addresses, and are obtained from the same STUN/TURN server for relayed and server reflexive candidates.

full: An Interactive Connectivity Establishment (ICE) implementation that adheres to the complete set of functionality described in [MS-ICE2].

Host Candidate: A **candidate** that is obtained by binding to ports on the local interfaces of the host computer. The local interfaces include both physical interfaces and logical interfaces such as Virtual Private Networks (VPNs).

ICE keep-alive message: A message that is sent periodically to keep active the **NAT bindings** at intermediate NATs and allocations on the **TURN server**.

initial offer: An offer that is sent by a **caller** and with the caller's **local candidates** when the caller initiates a media session with a **callee**.

Internet Protocol version 4 (IPv4): An Internet protocol that has 32-bit source and destination addresses. IPv4 is the predecessor of IPv6.

Internet Protocol version 6 (IPv6): A revised version of the Internet Protocol (IP) designed to address growth on the Internet. Improvements include a 128-bit IP address size, expanded routing capabilities, and support for **authentication** and privacy.

INVITE: A **Session Initiation Protocol (SIP)** method that is used to invite a user or a service to participate in a session.

Lite: An implementation that supports a minimal subset of Interactive Connectivity Establishment (ICE) functionality, as described in [MS-ICE2], to work with a Full ICE implementation. A Lite implementation responds to but does not send connectivity checks.

local candidate: A **candidate** whose transport addresses are local transport addresses.

local transport address: A transport address that is obtained by binding to a specific port from an IP address on the host computer. The IP address can be from physical interfaces or from logical interfaces such as Virtual Private Networks (VPNs).

NAT binding: The string representation of the protocol sequence, NetworkAddress, and optionally the endpoint. Also referred to as "string binding." For more information, see [C706] section "String Bindings."

network address translation (NAT): The process of converting between IP addresses used within an intranet, or other private network, and Internet IP addresses.

nominated: A **candidate pair** for which the nominated flag is set.

offer: A message that is sent by an offerer.

Ordinary Check: A connectivity check that is generated periodically by an **endpoint** based on the timers for connectivity checks.

peer: An additional **endpoint** that is associated with an endpoint in a session. An example of a peer is the **callee** endpoint for a **caller** endpoint.

peer-derived candidate: A **candidate** whose **transport addresses** are new mapping addresses, typically allocated by **NATs**, that are discovered during **connectivity checks**.

provisional answer: An optional message that carries **local candidates** for a **callee** and can be sent by the callee in response to a **caller's** initial offer.

Real-Time Transport Control Protocol (RTCP): A network transport protocol that enables monitoring of Real-Time Transport Protocol (RTP) data delivery and provides minimal control and identification functionality, as described in [RFC3550].

Real-Time Transport Protocol (RTP): A network transport protocol that provides end-to-end transport functions that are suitable for applications that transmit real-time data, such as audio and video, as described in [RFC3550].

Regular Nomination: The process of selecting a valid **candidate pair** for media flow by validating the candidate pairs with Simple Traversal of UDP through NAT (STUN) binding requests, and then selecting a valid candidate pair by sending STUN binding requests with a flag indicating that the candidate pair was nominated.

Relayed Candidate: A **candidate** that is allocated on the Traversal Using Relay NAT (TURN) server by sending an Allocate Request to the TURN server.

remote candidate: A **candidate** that belongs to a remote **endpoint** in a session.

remote endpoint: See **peer**.

RTCP packet: A control packet consisting of a fixed header part similar to that of RTP packets, followed by structured elements that vary depending upon the RTCP packet type. Typically, multiple RTCP packets are sent together as a compound RTCP packet in a single packet of the underlying protocol; this is enabled by the length field in the fixed header of each RTCP packet. See [RFC3550] section 3.

SDP offer: A **Session Description Protocol (SDP)** message that is sent by an offerer.

Server Reflexive Candidate: A **candidate** whose transport addresses is a **network address translation (NAT)** binding that is allocated on a NAT when an **endpoint** sends a packet through the NAT to the server. A Server Reflexive Candidate can be discovered by sending an allocate request to the **TURN server** or by sending a binding request to a **Simple Traversal of UDP through NAT (STUN)** server.

Session Description Protocol (SDP): A protocol that is used for session announcement, session invitation, and other forms of multimedia session initiation. For more information see [MS-SDP] and [RFC3264].

Session Initiation Protocol (SIP): An application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants. **SIP** is defined in [RFC3261].

Simple Traversal of UDP through NAT (STUN): A protocol that enables applications to discover the presence of and types of network address translations (NATs) and firewalls that exist between those applications and the Internet.

STUN candidate: A **candidate** whose transport addresses are STUN-derived transport addresses. See also **Simple Traversal of UDP through NAT (STUN)**.

Transmission Control Protocol (TCP): A protocol used with the Internet Protocol (IP) to send data in the form of message units between computers over the Internet. TCP handles keeping track of the individual units of data (called packets) that a message is divided into for efficient routing through the Internet.

transport address: A 3-tuple that consists of a port, an IPv4 or IPV6 address, and a transport protocol of User Datagram Protocol (UDP) or Transmission Control Protocol (TCP).

Traversal Using Relay NAT (TURN): A protocol that is used to allocate a public IP address and port on a globally reachable server for the purpose of relaying media from one **endpoint** to another **endpoint**.

triggered check: A connectivity check that is generated in response to a connectivity check packet that is received from a peer.

TURN candidate: A **candidate** whose transport addresses are TURN-derived transport addresses. See also **Traversal Using Relay NAT (TURN)**.

TURN server: An **endpoint** that receives **Traversal Using Relay NAT (TURN)** request messages and sends TURN response messages. The protocol server acts as a data relay, receiving data on the public address that is allocated to a protocol client and forwarding that data to the client.

User Datagram Protocol (UDP): The connectionless protocol within TCP/IP that corresponds to the transport layer in the ISO/OSI reference model.

Valid List: A list of **candidate pairs** that have been validated by **connectivity checks**.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as defined in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

Links to a document in the Microsoft Open Specifications library point to the correct section in the most recently published version of the referenced document. However, because individual documents in the library are not updated at the same time, the section numbers in the documents may not match. You can confirm the correct section numbering by checking the Errata.

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information.

[IETFDRAFT-ICENAT-19] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", draft-ietf-mmusic-ice-19, October 2007, http://tools.ietf.org/html/draft-ietf-mmusic-ice-19

[IETFDRAFT-ICETCP-07] Rosenberg, J., "TCP Candidates with Interactive Connectivity Establishment (ICE)", draft-ietf-mmusic-ice-tcp-07, July 2008, http://tools.ietf.org/html/draft-ietf-mmusic-ice-tcp-07

[IETFDRAFT-STUN-02] Rosenberg, J., Huitema, C., and Mahy, R., "Simple Traversal of UDP Through Network Address Translators (NAT) (STUN)", draft-ietf-behave-rfc3489bis-02, July 2005, http://tools.ietf.org/html/draft-ietf-behave-rfc3489bis-02

[MS-TURN] Microsoft Corporation, "Traversal Using Relay NAT (TURN) Extensions".

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

[RFC4571] Lazzaro, J., "Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport", RFC 4571, July 2006, http://www.ietf.org/rfc/rfc4571.txt

[RFC5389] J. Rosenberg, R. Mahy, P. Matthews, D. Wing, "Session Traversal Utilities for NAT (STUN)", http://tools.ietf.org/html/rfc5389

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

[RFC5766] R. Mahy, P. Matthews, J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", http://tools.ietf.org/html/rfc5766

1.2.2 Informative References

[MS-SDPEXT] Microsoft Corporation, "Session Description Protocol (SDP) Version 2.0 Extensions".

1.3 Overview

This protocol is used to establish media flow between a **callee endpoint** and a **caller** endpoint. In typical deployments, a **network address translation (NAT)** device or firewall might exist between the two endpoints that are intended to communicate. NATs and firewalls are deployed to provide private address space and to secure the private networks to which the endpoints belong. This type of deployment blocks incoming traffic. If the endpoint advertises its local interface address, the **remote endpoint** might not be able to reach it.

The address exposed by a NAT or firewall is not exactly what the endpoints need to determine the external routable mapping address created by the NAT, or the NAT-mapped address, for its local interface address. Moreover, NATs and firewalls exhibit differing behavior in the way they create the NAT-mapped addresses. ICE provides a generic mechanism to assist media in traversing NATs and firewalls without requiring the endpoints to be aware of their network topologies. ICE assists media in traversing NATs and firewalls by gathering one or more **transport addresses**, which the two endpoints (5) can potentially use to communicate, and then determining which transport address is best for both endpoints (5) to use to establish a media session.

The following figure shows a typical deployment scenario with two endpoints (5) that establish a media session.

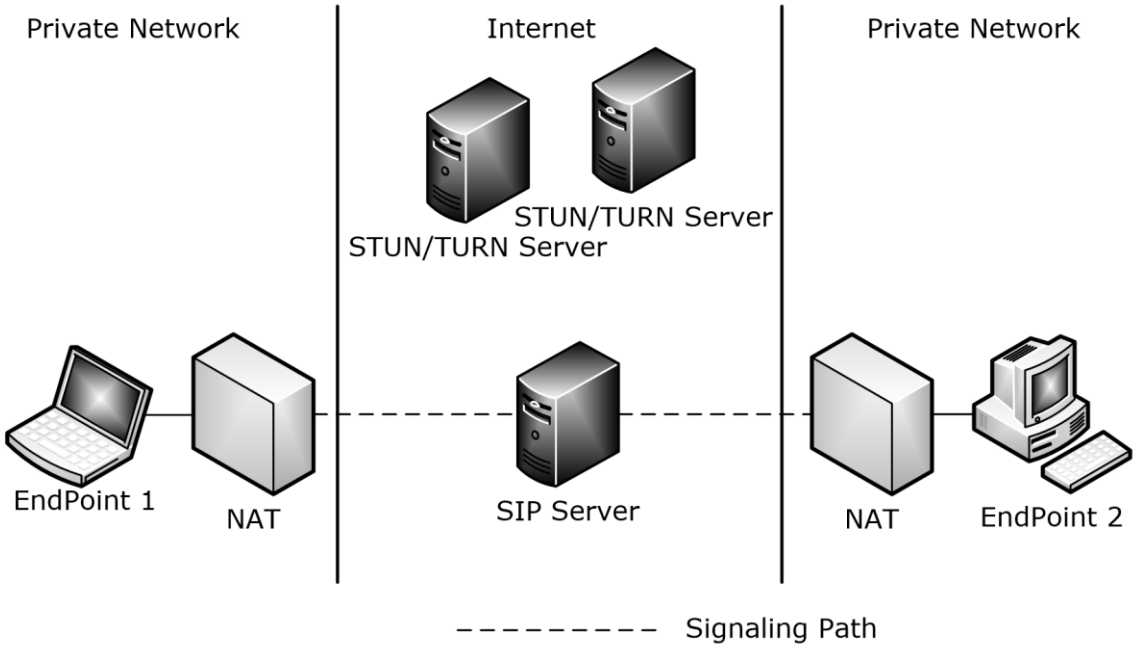


Figure 1: ICE deployment scenario

To facilitate ICE, a communication channel through which the endpoints can exchange messages, such as **Session Description Protocol (SDP)**, using a signaling protocol, such as **Session Initiation Protocol (SIP)**, is necessary. ICE assumes that such a channel exists and is not intended to be used for NAT traversal for these signaling protocols. ICE is often deployed in conjunction with **Simple Traversal of UDP through NAT (STUN)** and **Traversal Using Relay NAT (TURN)** servers. The endpoints can share the same STUN and **TURN servers** or use different servers.

The sequence diagram in the following figure outlines the various phases involved in establishing a session between two endpoints (5) using this protocol. These phases are:

1. **Candidates** gathering and the exchange of gathered transport addresses between the caller and callee endpoints.
2. **Connectivity checks**.
3. The exchange of candidates selected by the connectivity checks.

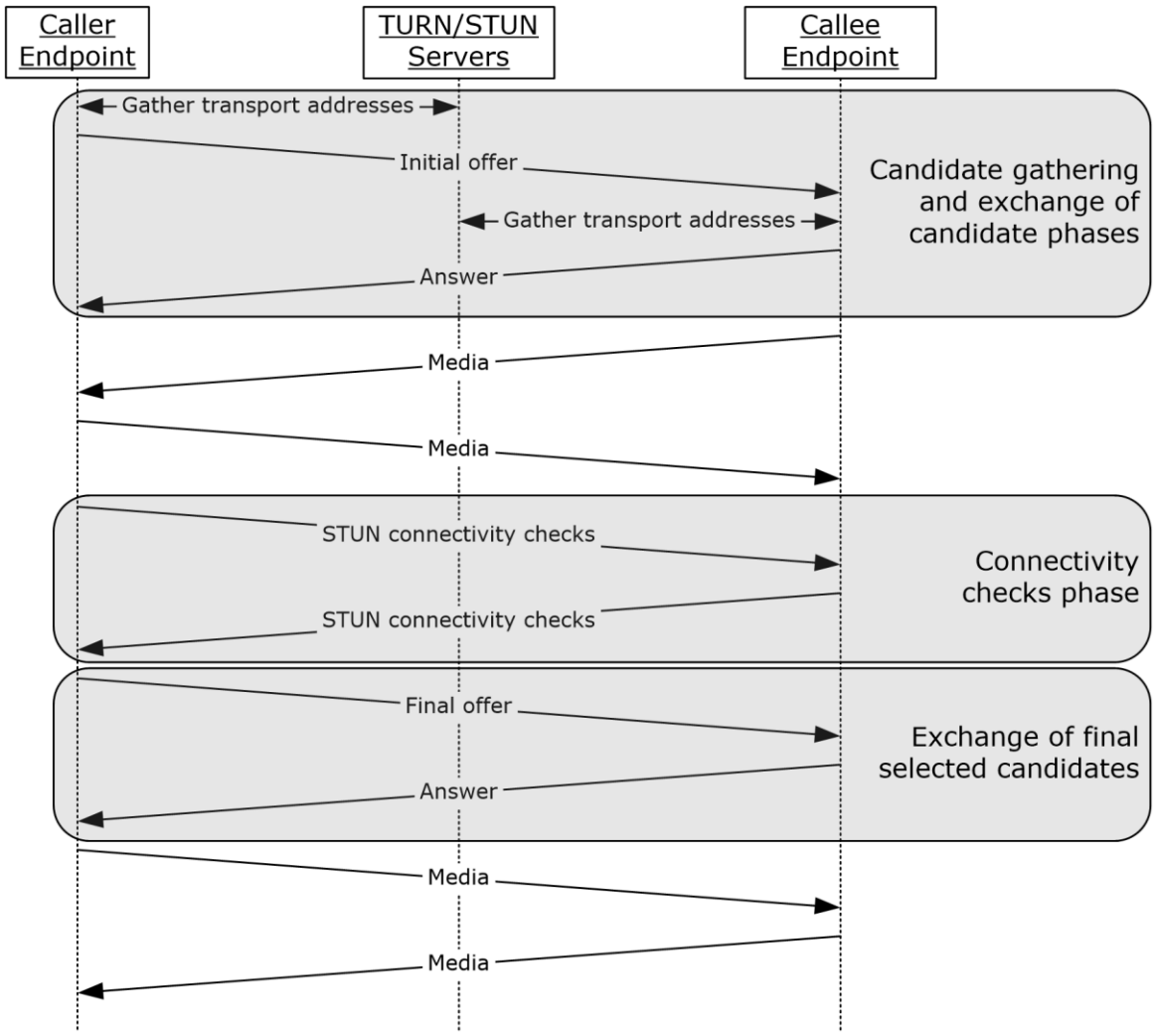


Figure 2: ICE sequence diagram

During the candidates gathering phase, the caller attempts to establish a media session and gathers transport addresses that can potentially be used to communicate with its **peer**. These potential transport addresses include:

- Transport addresses obtained by binding to attached network interfaces. These include both physical interfaces and virtual interfaces such as virtual private network (VPN), which is a **Host Candidate**.
- Transport addresses that are mappings on the public side of a NAT, which is a **Server Reflexive Candidate**.
- Transport addresses allocated from a TURN server, which is a **Relayed Candidate**.

The gathered transport addresses are used to form candidates. A candidate is a set of transport addresses that can potentially be used for media flow. For example, in the case of real-time media flow using **Real-Time Transport Protocol (RTP)**, each candidate consists of two **components**, one for RTP and another for **Real-Time Transport Control Protocol (RTCP)**.

Each gathered candidate is assigned a **foundation** and a priority value based on how they were obtained. This priority indicates the preference of an endpoint to use one candidate over another if

both candidates are reachable from the peer. The foundation is a string associated with each candidate. Two candidates have the same foundation if they are of the same type. Types of candidates are Host Candidates, Server Reflexive Candidates, Relayed Candidates, or **peer-derived candidates**. In addition to matching types, to have the same foundation the two candidates have the same **base** and are derived from the same STUN or TURN server. Candidates obtained from local network interfaces are often given a higher priority than the candidates obtained from TURN servers. The endpoint also designates one of the gathered candidates as the **default candidate** based on local policy.

The gathered candidates are then sent to the peer in the **offer**. The offer can be encoded into an **SDP offer** and exchanged over a signaling protocol such as SIP. The caller endpoint serves as the **controlling agent** and is responsible for selecting the final candidates for media flow.

The callee, after receiving the offer, follows the same procedure to gather its candidates. The gathered candidates are encoded and sent to the caller in the **answer**. With the exchange of candidates complete, both the endpoints are now aware of their peer's candidates.

The start of the connectivity checks phase is triggered at an endpoint when it is aware of its peer's candidates. Both endpoints pair up the **local candidates** and **remote candidates** to form a **Check List** of **candidate pairs** that are ordered based on the priorities of the candidate pairs. Each candidate pair consists of constituent component pairs and has the same foundation as the candidate pair. In the case of RTP, each candidate pair has an RTP component pair and an RTCP component pair. The candidate pair priorities are computed using the priorities of the local candidate and the remote candidate so that both endpoints have the same ordering of candidate pairs. Each candidate pair has an associated foundation that is formed as a concatenation of the foundations of the local candidate and the remote candidate that constitute the candidate pair. Candidate pairs with the same foundations have similar network properties, and this is leveraged to reduce the number of connectivity checks. If connectivity checks for a component pair fail, it is very likely that connectivity checks for other component pairs with the same foundation will also fail. Each endpoint goes through the candidate pair Check List and sets the state of the higher component pair, or the RTCP component pair, to a frozen state. If more than one candidate pair has the same foundation, all candidate pairs except for the highest priority candidate pair with the same foundation are set to a frozen state. When the connectivity check for a component pair succeeds, all component pairs with the same foundations are unfrozen. The callee serves as the **controlled agent** and waits for the controlling agent to select the final candidate pair for media flow.

Both endpoints systematically perform connectivity checks, starting from the top of the candidate pair Check List to determine the highest priority candidate pair that can be used by the endpoints for establishing a media session. Connectivity checks involve sending peer-to-peer STUN binding request messages and responses from the **local transport addresses** to the remote transport addresses of each candidate pair in the list. Once a STUN binding request message is received, and it generates a successful STUN binding response message for a component pair, the component pair is considered to be in successful state.

The endpoints can begin streaming media from the local default candidate to the remote default candidate after the exchange of candidates is finished, even before the **default candidate pair** is validated by connectivity checks, but there is no guarantee that the media will reach the peer during this time.

The connectivity checks for the candidate pairs are spaced at regular intervals to avoid flooding the network. Depending on the topology, many of the candidate pairs might fail connectivity checks. For example, in the topology illustrated in the preceding figure titled "ICE deployment scenario", the transport addresses obtained from the local network interfaces cannot be used directly to establish a connection, because both endpoints are behind NATs. These connectivity checks, sent periodically to validate the candidate pairs, are called **Ordinary Checks**. In addition, to optimize the connectivity checks, an endpoint, on receiving a STUN binding request for a candidate pair, immediately schedules a connectivity check for that candidate pair. These connectivity checks are called **triggered checks**.

The endpoints can also discover new candidates during the connectivity check phase. This can happen in either of two scenarios:

- The STUN binding request message is received from a transport address that does not match any of the remote candidates.
- The STUN binding response message has a mapped address that does not match the transport address of any of the local candidates.

These scenarios arise if new external mappings are created by the NATs residing between the endpoints. Connectivity checks are sent out on candidate pairs formed using these newly created candidates. These candidates can potentially be used for media flow as well.

The controlling agent concludes the connectivity checks by nominating a valid candidate pair found by the connectivity checks for media flow. The controlling agent can follow either **Regular Nomination** or **Aggressive Nomination** to nominate the validated candidate pairs. If the controlling agent is following Regular Nomination, it allows connectivity checks to continue until at least one valid candidate pair has been found. At the end of the connectivity checks, the controlling agent picks the best valid candidate pair from the **Valid List** and sends another round of STUN binding requests for this candidate pair with a flag set to notify the peer that this candidate pair has been **nominated** for media flow. In the case of Aggressive Nomination, the controlling agent sets this flag on every STUN binding request. With Aggressive Nomination, the ICE processing completes when connectivity checks succeed for the first candidate pair, and the controlling agent does not have to send a second STUN binding request to nominate the candidate pair. Aggressive Nomination is faster than Regular Nomination but does not always select the optimal path that has the lowest latency. At the end of the connectivity checks phase, the controlling agent sends a **final offer** with only the best local and remote candidate selected during the connectivity checks phase. The peer acknowledges the final offer with an answer, and both endpoints begin using the selected candidate pair for media flow.

1.4 Relationship to Other Protocols

This protocol is an application layer protocol that depends on, and works with, the **Transmission Control Protocol (TCP)** and **User Datagram Protocol (UDP)** transport protocols for **Internet Protocol version 4 (IPv4)** / **Internet Protocol version 6 (IPv6)** addresses only.

This protocol works with implementations of **Traversal Using Relay NAT (TURN)** protocols, as described in [MS-TURN], to create **TURN candidates** and **STUN candidates**.

This protocol can perform **connectivity checks** only with **endpoints** that follow the message formats in the **Simple Traversal of UDP through NAT (STUN)** specifications and that follow the STUN attributes and usage specification in section 3.1.4.3.

This protocol depends on signaling protocols, such as **Session Initiation Protocol (SIP)**, to perform an **offer** and **answer** exchange of encoded messages, such as **Session Description Protocol (SDP)** messages as described in [MS-SDPEXT].

This protocol is used to establish a communication channel that is eventually used for media flow for protocols such as **Real-Time Transport Protocol (RTP)** and **Real-Time Transport Control Protocol (RTCP)**.

1.5 Prerequisites/Preconditions

This protocol requires that the **endpoints** are able to communicate through a signaling protocol, such as **Session Initiation Protocol (SIP)**, to exchange **candidates**.

1.6 Applicability Statement

This protocol is a **full** implementation, and requires the **peer endpoint** to perform **Regular Nomination**. It does not support or work with peer endpoints that perform **Aggressive Nomination**.

This protocol treats a **Lite** implementation peer as a peer that does not support ICE and does not follow the procedures for handling a Lite implementation peer.

This protocol treats each stream in a session independently for ICE processing, if the session has more than one stream. The procedures specified in this protocol are per media stream.

This protocol does not support ICE restarts.

This protocol requires **TURN servers** to be deployed to facilitate communication across **NAT** devices and firewalls. In the absence of TURN servers, this protocol might not be able to establish connectivity between endpoints in such topologies.

This protocol is appropriate for establishing a communication channel between two endpoints for media exchange.

This protocol can operate in two modes: regular and **Transmission Control Protocol (TCP)** only. This protocol cannot be used for establishing a communication channel through TCP in the absence of a TURN server in regular mode. Both the **caller** and **callee** endpoints need to support and operate in the same mode for this protocol to establish connectivity.

This protocol is used to establish connectivity for streaming **Real-Time Transport Protocol (RTP)** media. As a result, this protocol supports exactly two **components** for each **candidate**. It does not support scenarios that require less than two or greater than two components for each candidate.

This protocol does not guarantee consecutive ports for RTP and **Real-Time Transport Control Protocol (RTCP)**. As a result, endpoints that need to communicate with an endpoint that implements this protocol are required to support sending and receiving media to RTP and RTCP on nonconsecutive ports, whether or not they support ICE itself.

This protocol multiplexes both components to the same IP address and port when the connection is established through TCP. The application layer is required to demultiplex the data sent for the two components if TCP candidates are used. For example, if the two components are RTP and RTCP, both RTP and RTCP are delivered to the same IP address and port. Both endpoints multiplex components over TCP.

This protocol supports the multiplexing of RTP and RTCP components to the same IP address and port when the connection is established over **User Datagram Protocol (UDP)** where multiplexing support is negotiated as described in [RFC5761].

During the **connectivity checks**, **ICE keep-alive messages** are sent for both RTP and RTCP components for validated component pairs and for **candidate pairs** whose **local candidates** are **Relayed Candidates**. For the candidate that is being used for media flow, the ICE keep-alive messages are sent only for the RTP component's **transport addresses**. **RTCP packets** are sent to keep the **NAT bindings** and **Traversal Using Relay NAT (TURN)** allocations active for the RTCP component's transport addresses. ICE keep-alive messages are sent regardless of whether UDP or TCP is the underlying transport.

1.7 Versioning and Capability Negotiation

This protocol is implemented on top of the **Transmission Control Protocol (TCP)** and **User Datagram Protocol (UDP)** transport protocols for **Internet Protocol version 4 (IPv4)**/**Internet Protocol version 6 (IPv6)** as described in section 2.1.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

None.

2 Messages

2.1 Transport

This protocol uses the **Transmission Control Protocol (TCP)** and **User Datagram Protocol (UDP)** transport protocols for **Internet Protocol version 4 (IPv4)**/**Internet Protocol version 6 (IPv6) endpoints**.<1>

Applications implementing this protocol MUST NOT send messages that are greater than 1,500 bytes in length, and MUST be able to receive messages of 1,500 bytes or less in length.

2.2 Message Syntax

This section specifies the various messages used by the implementation of this protocol. This includes both outgoing and incoming messages. This protocol does not define its own custom message formats. The messages used by this protocol, and the protocols they belong to, are listed later in this section.

2.2.1 TURN Messages

This protocol SHOULD use a **TURN server** that implements a protocol, as specified in [MS-TURN], to discover **Server Reflexive Candidates** and **Relayed Candidates**. The **endpoint** implementing that protocol to communicate with the TURN server MUST use the message syntax that is specified in [MS-TURN] section 2.

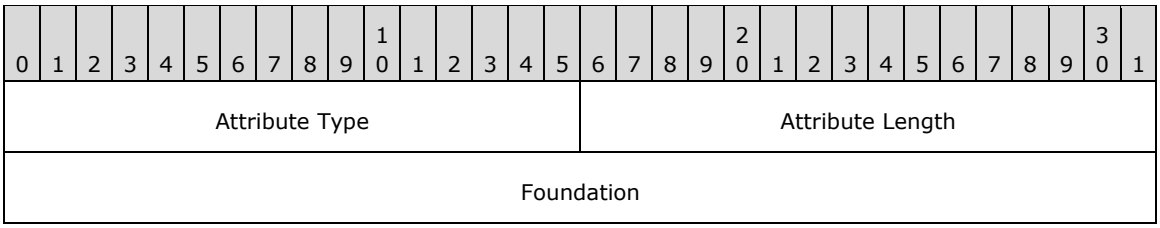
2.2.2 STUN Messages

This protocol uses **Simple Traversal of UDP through NAT (STUN)** binding request and response messages for **connectivity checks** between the two **endpoints**. The protocol supports message formats specified in [IETFDRAFT-STUN-02] section 10 and [RFC5389] section 11<2>. The message format is negotiated according to section 3.1.5.2. STUN messages sent over **Transmission Control Protocol (TCP)** MUST follow the framing method specified in [RFC4571] section 2. This method is required to demultiplex the received application data and STUN packets. STUN messages MUST support the STUN extensions and attributes specified in [IETFDRAFT-ICENAT-19] section 19. The **XOR-MAPPED-ADDRESS** attribute MUST have a value of 0x0020.

This protocol defines two additional attributes: **CANDIDATE-IDENTIFIER** and **IMPLEMENTATION-VERSION**, which MUST be supported per the procedures in [IETFDRAFT-STUN-02] section 10.2 if the message format follows [IETFDRAFT-STUN-02], or in [RFC5389] section 15 if the message format follows [RFC5389]. The **CANDIDATE-IDENTIFIER** attribute MUST be sent only with STUN binding request messages. The **IMPLEMENTATION-VERSION** attribute MUST be added to all STUN binding request and response messages.

2.2.2.1 CANDIDATE-IDENTIFIER

The **CANDIDATE-IDENTIFIER** attribute MUST be added to **Simple Traversal of UDP through NAT (STUN)** binding request messages that are sent for **connectivity checks**. The **CANDIDATE-IDENTIFIER** attribute is used to identify the **remote candidate** from which the connectivity check is received. The value of **CANDIDATE-IDENTIFIER** MUST be a valid **foundation** string. If the length of the **CANDIDATE-IDENTIFIER** value is not at a 4-byte boundary, the value MUST be padded with NULLs to be at a 4-byte boundary on the wire. The usage of this attribute MUST follow the specification in section 3.1.4.8.2.4.



Attribute Type (2 bytes): The type of the attribute. The value of this field MUST be 0x8054.

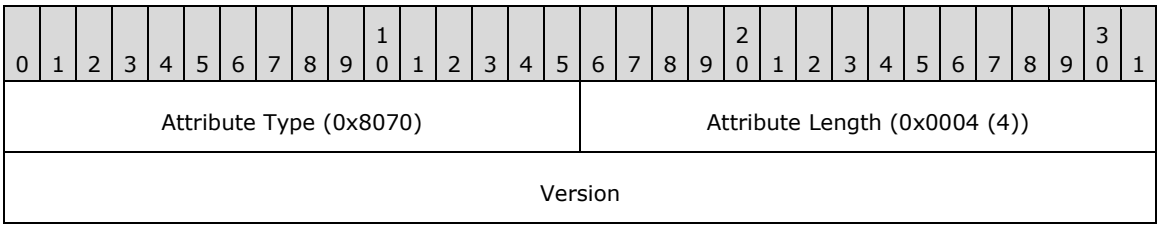
Attribute Length (2 bytes): The length of the attribute.

Foundation (4 bytes): The foundation. The value of this field MUST be set to the foundation of the **local candidate** for which the request is being sent, if the **candidate** is not a **peer-derived candidate**. If the local candidate is a peer-derived candidate, the value MUST be set to the foundation of the peer-derived local candidate's **base**.

2.2.2.2 IMPLEMENTATION-VERSION

This section follows the behavior described in endnote <3>.

The **IMPLEMENTATION-VERSION** attribute is the ICE protocol implementation version. This attribute SHOULD be included in all **connectivity check** request and response messages. The format of this attribute is as follows.



Attribute Type (2 bytes): 0x8070 specifies the type of the attribute.

Attribute Length (2 bytes): 0x0004 (4) specifies the length of the attribute.

Version (4 bytes): The version number, which an ICE implementation MUST<4> set.

2.2.3 ICE keep-alive

The **ICE keep-alive message** MUST be a valid **Simple Traversal of UDP through NAT (STUN)** binding request message, as specified in [IETFDRAFT-STUN-02] section 8.1, and MUST follow the additional specifications in this section. ICE keep-alive messages sent over **Transmission Control Protocol (TCP)** MUST follow the framing method specified in [RFC4571] section 2. The **transaction ID** can be any valid **transaction ID**. The ICE keep-alive message MUST have the **MESSAGE-INTEGRITY** attribute set to a value of 0 or a valid message integrity value. The ICE keep-alive message MUST NOT have any other attributes.

3 Protocol Details

3.1 Common Details

The procedures specified apply to both the **Transmission Control Protocol (TCP)** and **User Datagram Protocol (UDP)** transport protocols unless a procedure explicitly specifies a transport protocol. This protocol MUST support operating in either the regular mode or the TCP-only mode, based on the cue from the application layer that builds on top of this protocol. By default, this protocol operates in the regular mode. The differences between the operating modes exist only during the **candidates** gathering phase, as specified in section 3.1.4.8.1.

3.1.1 Abstract Data Model

This section describes a conceptual model of possible data organization that an implementation maintains to participate in this protocol. The described organization is provided to facilitate the explanation of how the protocol behaves. This document does not mandate that implementations adhere to this model as long as their external behavior is consistent with that described in this document.

This protocol uses the abstract model specified in [IETFDRAFT-ICENAT-19] section 7.

3.1.2 Timers

The **Candidates Gathering Phase** timer tracks the maximum duration for the **candidates** gathering phase. This timer MUST have a default value of 10 seconds.

The **Connectivity Checks Phase** timer tracks the maximum duration for which **connectivity checks** can be performed between the **candidate pairs**. The maximum timeout for this timer MUST be set to 10 seconds.

The **ICE keep-alive** timer tracks the spacing of **ICE keep-alive messages**. These messages are sent to keep the **NAT bindings** and **Traversal Using Relay NAT (TURN)** allocations active. This timer MUST have a default value of 19 seconds or less.

The **USE-CANDIDATE Checks** timer tracks the maximum duration for which **USE-CANDIDATE** checks can be performed to nominate the candidate pairs selected by connectivity checks as part of **Regular Nomination**. This timer is applicable only for the **controlling agent**. The maximum timeout for this timer SHOULD be 10 seconds.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

This section outlines the higher-layer events that trigger the start of the various phases of this protocol for connection establishment. Updating **candidate** lists during and after the **connectivity checks** is allowed, as specified in [IETFDRAFT-ICENAT-19] section 9.3.1.4. This protocol specifies that there MUST NOT be an additional **offer** or exchange of candidates other than those specified in this section. Processing for this protocol is specified for each media stream. If connectivity has to be established for more than one media stream, connectivity establishment MUST be carried out independently for each media stream. If the **transport address** for media or any of the candidates needs to change, the **endpoints** MUST stop the specific media stream and restart it so that the procedure outlined in this section is triggered again. In case the **peer** does not support Interactive Connectivity Establishment (ICE), the default transport addresses used for media MUST NOT be changed after the **initial offer** and **answer**.

3.1.4.1 Sending the Initial Offer

The **caller** attempting to establish a media session with a **peer** MUST gather its **local candidates** as specified in section 3.1.4.8.1. After the **candidates** are gathered, they MUST be encoded before being sent to the peer **endpoint** through the pre-established signaling channel. For example, the candidates can be encoded into an **SDP offer**.

The caller MUST designate one of the local candidates as the **default candidate** in the **initial offer**. In regular mode, the default candidate MUST be a **User Datagram Protocol (UDP)** candidate. If no UDP candidate has been gathered, the call MUST fail. In TCP-only mode, the default candidate MUST be a **Transmission Control Protocol (TCP)** candidate, and no UDP candidates can be gathered or sent in the **offer**. If no TCP candidate has been allocated, the call MUST fail. After the candidates have been gathered successfully, the caller SHOULD be ready to respond to **connectivity checks** from the **callee**.

3.1.4.2 Receiving the Initial Offer and Generating the Answer

The **callee**, on receiving the **initial offer**, MUST gather its **local candidates** as specified in section 3.1.4.8.1. After the **candidates** are gathered, they MUST be encoded before being sent to the **peer** through the pre-established signaling channel. For example, the candidates can be encoded into an SDP **answer**.

The callee MUST designate one of the local candidates as the **default candidate** in the answer to the initial offer. In regular mode, the default candidate MUST be a **User Datagram Protocol (UDP)** candidate. If no UDP candidates are gathered, the call MUST fail. In TCP-only mode, the default candidate MUST be a **Transmission Control Protocol (TCP)** candidate, and no UDP candidates can be gathered or sent in the answer. If no TCP candidate is gathered, the call MUST fail.

When the callee receives the initial offer with the **caller's** candidates, the callee MUST begin the **connectivity checks** phase, as specified in section 3.1.4.8.2, after gathering its local candidates. Applications that require reducing the perceived latency of call establishment for the user SHOULD have the callee encode the gathered candidates and send them in a **provisional answer** to the caller before sending the answer to the initial offer. If an **endpoint** sends a provisional answer, the subsequent answer for the initial offer MUST have the same set of candidates and default candidate as the provisional answer.

3.1.4.3 Processing the Provisional Answer to the Initial Offer

The **caller**, after receiving the **provisional answer** with the **callee's candidates**, MUST begin the **connectivity checks** as specified in section 3.1.4.8.2. A single **initial offer** can result in multiple provisional answers being received as a result of forking. The Interactive Connectivity Establishment (ICE) processing MUST be carried out independently for each provisional answer, as specified in [IETFDRAFT-ICENAT-19] section 6.

Implementations of this protocol SHOULD NOT support the processing of more than 20 provisional answers. Implementations of this protocol can support less than 20 provisional answers if the resources are not available to process 20 provisional answers. Provisional answers that arrive after the maximum number of supported provisional answers has been exceeded MUST be ignored.

3.1.4.4 Processing the Answer to the Initial Offer from a Full ICE Peer

The **caller**, upon receiving the **answer** to its **initial offer** with the **callee's candidates**, MUST begin the **connectivity checks** phase, as specified in section 3.1.4.8.2, if the connectivity checks were not already started as a result of receiving a **provisional answer**. If a provisional answer was already received from the **peer endpoint**, connectivity checks that were started as a result of processing the provisional answer MUST be continued.

3.1.4.4.1 Processing the Answer to the Initial Offer from a Peer that Does Not Support ICE or that Supports a Lite Implementation

If an **answer** is received from a **peer** that does not support Interactive Connectivity Establishment (ICE) or that supports a **Lite** implementation, the procedure outlined in this section MUST be followed.<5>

Simple Traversal of UDP through NAT (STUN) binding request messages MUST be sent by the **caller** from the **default candidate** to the **transport addresses**, one for **Real-Time Transport Protocol (RTP)** and one for **Real-Time Transport Control Protocol (RTCP)**, advertised by the peer that does not support ICE.

These STUN binding request messages serve only to open permissions on the **TURN servers** and **NAT** devices for the peer that does not support ICE. After the answer is received from a peer that does not support ICE or that supports a Lite implementation, no further **connectivity checks** processing or **offer** and answer exchanges are required. The default candidate advertised in the **initial offer** MUST be used for media flow to the **remote candidate** advertised in the answer.

3.1.4.5 Generating the Final Offer

At the end of the **connectivity checks** phase, the **controlling agent** MUST send the **final offer**. The final offer MUST be encoded and MUST contain only the **local candidate** and **remote candidate** selected by this protocol, to its **peer**. For example, the final offer can be encoded into an **SDP offer**.

The final offer MUST be generated, even if the selected local and remote candidates match the default local and remote candidates, respectively, of the **initial offer** and **answer**.

3.1.4.6 Receiving the Final Offer and Generating the Answer

The **controlled agent**, upon receiving the **final offer**, MUST validate the **candidates** received in the final offer by verifying that it has a **candidate pair** that consists of the local and **remote candidates** in the final offer. If the remote candidate in the final offer is not known, the call MUST fail. If the **local candidate** in the final offer is not known, the **endpoint** checks the **triggered check** queue to see if there are triggered checks queued as a result of the **Simple Traversal of UDP through NAT (STUN)** binding request with the **nomination** flag received from the **controlling agent** during nomination. If no corresponding triggered checks are found, the call MUST fail. If found, these triggered checks are processed until either the local candidate that matches the local candidate in the final offer is discovered or the application layer terminates the call.

If a matching candidate pair for the candidates in the final offer is found, the endpoint MUST switch to using the local and remote candidates in the **offer** for media flow. It MUST acknowledge the receipt of the final offer similarly, with a response that MUST contain only the local candidate and the remote candidate to be used for media flow. If the selected local candidate is a **TURN candidate**, a **Set Active Destination** message, as specified in [MS-TURN] section 3.2.5, SHOULD be sent for that candidate, and the subsequent processing SHOULD also be as specified in [MS-TURN] section 3.2.5. Local candidates other than the selected local candidate SHOULD be freed.

3.1.4.7 Processing the Answer to the Final Offer

The **controlling agent**, after receiving the **answer** to its **final offer**, MUST validate the **local candidate** and **remote candidate** in the answer to ensure that they are the same **candidates** that the controlling agent selected and sent in the final offer. If the validation fails, the call MUST fail. If the answer is successfully validated, the controlling agent MUST switch to using the local and remote candidates in the answer for media flow. An **endpoint**, on receiving the answer to its final offer, SHOULD free all local candidates other than the selected local candidate. If the selected local candidate is a **TURN candidate**, a **Set Active Destination** message, as specified in [MS-TURN] section 3.2.5, SHOULD be sent for that candidate.

3.1.4.8 Common Procedures

The following sections specify common procedures triggered by higher-layer events.

3.1.4.8.1 Candidates Gathering Phase

The **candidates** gathering phase is common to both the **caller** and **callee**. Sections 3.1.4.1 and 3.1.4.2 specify when the candidates gathering phase is triggered on caller and callee **endpoints**. This section specifies the operations involved in the candidates gathering phase. The candidates gathering phase MUST end when the **Candidates Gathering Phase** timer fires or when the process of gathering candidates is complete.

Because this protocol is used for streaming **Real-Time Transport Protocol (RTP)** media, each candidate MUST have two **components**. One component is for RTP; the other is for **Real-Time Transport Control Protocol (RTCP)**. This protocol gathers **Internet Protocol version 4 (IPv4)**/**Internet Protocol version 6 (IPv6)** addresses for **Transmission Control Protocol (TCP)** and **User Datagram Protocol (UDP)** transports as specified in [IETFDRAFT-ICENAT-19] section 4.1.1. Applications can choose to gather IPv4 candidates only or IPv6 candidates only or both during the candidates gathering phase.<6>

Implementers of this protocol MUST NOT support sending<7> more than 40 candidates in the **offer** or **answer**. If an endpoint gathers more than 40 candidates, it MUST send no more than 40 candidates for the offer exchange and discard the additional candidates. This is done to mitigate the **Simple Traversal of UDP through NAT (STUN)** amplification attack specified in section 5.1.4.

This protocol does not implement candidate ICE keep-alive messages, as specified in [IETFDRAFT-ICENAT-19] section 4.1.1.4. At the end of the candidates gathering phase, redundant candidates MUST be eliminated, as specified in [IETFDRAFT-ICENAT-19] section 4.1.3. The **default candidates** MUST be selected, as specified in sections 3.1.4.1 and 3.1.4.2.

3.1.4.8.1.1 Gathering Candidates

This section specifies the **candidate** types and behavior supported by this protocol. An implementer of this protocol MUST support gathering candidates of the following types:

- **User Datagram Protocol (UDP) Host Candidates**
- UDP **Server Reflexive Candidates**
- UDP **Relayed Candidates**
- Active/passive **Transmission Control Protocol (TCP)** Host Candidates
- Active TCP Server Reflexive Candidates
- Active/passive TCP Relayed Candidates

The implementer of this protocol MUST NOT support the gathering of other candidate types or candidate behaviors. The **Real-Time Transport Protocol (RTP)** and **Real-Time Transport Control Protocol (RTCP) components** of UDP candidates MUST have the same IP address, and different ports. For TCP candidates, both components MUST have the same IP address and port. As a result, both of the components of the TCP candidates MUST be multiplexed onto the same IP address and port. For more details, see [RFC5761]. If both **Internet Protocol version 4 (IPv4)** and **Internet Protocol version 6 (IPv6)** candidates of the same type are available, the IPv4 candidate SHOULD be given a higher priority than the IPv6 candidate.<8>

The gathered **transport addresses** MUST NOT be NULL, multicast, broadcast or link-local IP addresses. The ports of the gathered transport addresses MUST NOT be in the port range 0–1023.

3.1.4.8.1.2 Gathering UDP Candidates

User Datagram Protocol (UDP) local candidates are obtained by binding to ephemeral ports on all available network interfaces. This includes both physical interfaces and virtual interfaces, such as virtual private network (VPN). The **candidates** MUST be gathered as specified in [IETFDRAFT-ICENAT-19] section 4.1.

UDP **Relayed Candidates** SHOULD be obtained following the procedures for allocating candidates on the **TURN server** as specified in [MS-TURN] section 3.2.4.1.

UDP **Server Reflexive Candidates** SHOULD be discovered by following the procedure specified in [MS-TURN] section 3.2.5.1.

Implementations of this protocol SHOULD NOT pair all **Host Candidates** with the TURN server, as specified in [IETFDRAFT-ICENAT-19] section 4.1.1.2. This protocol selects the best host interface to communicate with a configured TURN server and gathers Server Reflexive Candidates and Relayed Candidates only for that interface.

3.1.4.8.1.3 Gathering TCP Candidates

The gathering of **Transmission Control Protocol (TCP) candidates** varies based on the operation mode. The following subsections specify the differences in candidate gathering between the two operation modes. This protocol does not support gathering simultaneous-open candidates and does not work with simultaneous-open candidates. A simultaneous-open candidate is one for which the agent will attempt to open a connection simultaneously with its peer.

Implementations of this protocol MUST set the ports to any value in the valid port range, which is outside of 0–1023. The port value advertised is not important because the outbound connection for the active candidates is done from ephemeral ports. Implementations of this protocol MUST multiplex both **Real-Time Transport Protocol (RTP)** and **Real-Time Transport Control Protocol (RTCP)** on the same port.

3.1.4.8.1.3.1 TCP-Only Mode

In the TCP-only mode of operation, one active and one passive **Host Candidate** MUST be gathered from every available network interface.

Transmission Control Protocol (TCP) candidates SHOULD be obtained following the procedures for allocating candidates on the **TURN server**, as specified in [MS-TURN] section 3.2.4.1. If multiple local interfaces are available, **Relayed Candidates** and **Server Reflexive Candidates** SHOULD be obtained by selecting the best local interface to communicate with the relay.

TCP Server Reflexive Candidates SHOULD be discovered by following the procedure specified in [MS-TURN] section 3.2.5.1.

Each TCP Relayed Candidate gathered serves as both an active and a passive candidate and MUST be advertised separately as an active Relayed Candidate and as a passive Relayed Candidate in the encoded offer when the candidates are exchanged. The Server Reflexive Candidate obtained from the allocate response SHOULD be advertised as an active Server Reflexive Candidate.

3.1.4.8.1.3.2 Regular Mode

In regular mode, active and passive **Host Candidates** are not gathered. Only **Relayed Candidates** and **Server Reflexive Candidates** SHOULD be gathered if **TURN servers** have been configured. The procedures for gathering Server Reflexive Candidates and Relayed Candidates is the same as that specified in section 3.1.4.8.1.3.1.

When no **Transmission Control Protocol (TCP)** TURN servers have been configured in regular mode, implementations of this protocol SHOULD create an active Server Reflexive Candidate that has the same IP address as one of the **User Datagram Protocol (UDP)** Server Reflexive Candidates, if one exists. If no UDP Server Reflexive Candidates exist, a Server Reflexive Candidate SHOULD be

created with the same IP address as one of the host UDP **candidates**. This is done to facilitate a potential TCP connectivity path, even in the absence of TCP Relayed Candidates for one of the **endpoints** in regular operation mode.

3.1.4.8.1.4 Generating Candidate Foundations and Priorities

The **candidate foundations** MUST be generated as specified in [IETFDRAFT-ICENAT-19] section 4.1.1.3. Priorities for **User Datagram Protocol (UDP)** and **Transmission Control Protocol (TCP)** candidates MUST be computed as specified in [IETFDRAFT-ICENAT-19] section 4.1.2 and [IETFDRAFT-ICETCP-07] section 3.2, respectively.

3.1.4.8.2 Connectivity Checks Phase

An application triggers the start of the **connectivity checks** phase after the completion of the **offer** and **answer** exchange of **candidates**, as specified in sections 3.1.4.2, 3.1.4.3, and 3.1.4.4. The connectivity checks phase MUST have an overall worst case timeout, as specified in section 3.1.6.2. When a connectivity check request and a connectivity check response packet have been received from the **peer**, the timeout for the connectivity check MUST be reduced to the value specified in section 3.1.6.2.

During the connectivity checks phase, whenever a connectivity check request or response is sent, an additional connectivity check request or response SHOULD<9> be sent along with it. This additional request or response is identical to the original request or response, except that the fingerprint for these additional messages MUST be computed through a **cyclic redundancy check (CRC)** using the following lookup table of hexadecimal values. This is done to interoperate with implementations that used the CRC lookup table.

Column 0	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7
00000000	77073096	EE0E612C	990951BA	076DC419	706AF48F	E963A535	9E6495A3
0EDB8832	79DCB8A4	E0D5E91E	97D2D988	09B64C2B	7EB17CBD	E7B82D07	90BF1D91
1DB71064	6AB020F2	F3B97148	84BE41DE	1ADAD47D	6DDDE4EB	F4D4B551	83D385C7
136C9856	646BA8C0	FD62F97A	8A65C9EC	14015C4F	63066CD9	FA0F3D63	8D080DF5
3B6E20C8	4C69105E	D56041E4	A2677172	3C03E4D1	4B04D447	D20D85FD	A50AB56B
35B5A8FA	42B2986C	DBBBC9D6	ACBCF940	32D86CE3	45DF5C75	DCD60DCF	ABD13D59
26D930AC	51DE003A	C8D75180	BFD06116	21B4F4B5	56B3C423	CFBA9599	B8BDA50F
2802B89E	5F058808	C60CD9B2	B10BE924	2F6F7C87	58684C11	C1611DAB	B6662D3D
76DC4190	01DB7106	98D220BC	EFD5102A	71B18589	06B6B51F	9FBFE4A5	E8B8D433
7807C9A2	0F00F934	9609A88E	E10E9818	7F6A0DBB	086D3D2D	91646C97	E6635C01
6B6B51F4	1C6C6162	856530D8	F262004E	6C0695ED	1B01A57B	8208F4C1	F50FC457
65B0D9C6	12B7E950	8BBE8EA	FCB9887C	62DD1DDF	15DA2D49	8CD37CF3	FBD44C65
4DB26158	3AB551CE	A3BC0074	D4BB30E2	4ADFA541	3DD895D7	A4D1C46D	D3D6F4FB
4369E96A	346ED9FC	AD678846	DA60B8D0	44042D73	33031DE5	AA0A4C5F	DD0D7CC9
5005713C	270241AA	BE0B1010	C90C2086	5768B525	206F85B3	B966D409	CE61E49F
5EDEF90E	29D9C998	B0D09822	C7D7A8B4	59B33D17	2EB40D81	B7BD5C3B	C0BA6CAD

Column 0	Column 1	Column 2	Column 3	Column 4	Column 5	Column 6	Column 7
EDB88320	9ABFB3B6	03B6E20C	74B1D29A	EAD54739	9DD277AF	04DB2615	73DC1683
E3630B12	94643B84	0D6D6A3E	7A6A5AA8	E40ECF0B	9309FF9D	0A00AE27	7D079EB1
F00F9344	8708A3D2	1E01F268	6906C2FE	F762575D	806567CB	196C3671	6E6B06E7
FED41B76	89D32BE0	10DA7A5A	67DD4ACC	F9B9DF6F	8EBEEFF9	17B7BE43	60B08ED5
D6D6A3E8	A1D1937E	38D8C2C4	4FDFF252	D1BB67F1	A6BC5767	3FB506DD	48B2364B
D80D2BDA	AF0A1B4C	36034AF6	41047A60	DF60EFC3	A867DF55	316E8EEF	4669BE79
CB61B38C	BC66831A	256FD2A0	5268E236	CC0C7795	BB0B4703	220216B9	5505262F
C5BA3BBE	B2BD0B28	2BB45A92	5CB36A04	C2D7FFA7	B5D0CF31	2CD99E8B	5BDEAE1D
9B64C2B0	EC63F226	756AA39C	026D930A	9C0906A9	EB0E363F	72076785	05005713
95BF4A82	E2B87A14	7BB12BAE	0CB61B38	92D28E9B	E5D5BE0D	7CDCEFB7	0BDBDF21
86D3D2D4	F1D4E242	68DDB3F8	1FDA836E	81BE16CD	F6B9265B	6FB077E1	18B74777
88085AE6	FF0F6A70	66063BCA	11010B5C	8F659EFF	F862AE69	616BFFD3	166CCF45
A00AE278	D70DD2EE	4E048354	3903B3C2	A7672661	D06016F7	4969474D	3E6E77DB
AED16A4A	D9D65ADC	40DF0B66	37D83BF0	A9BCAE53	DEBB9EC5	47B2CF7F	30B5FFE9
BDBDF21C	CABAC28A	53B39330	24B4A3A6	BAD03605	CDD70693	54DE5729	23D967BF
B3667A2E	C4614AB8	5D681B02	2A6F2B94	B40BBE37	C30C8EA1	5A05DF1B	2D02EF8D

The transmission of this additional connectivity check packet SHOULD be stopped on receiving a connectivity check request or response from the peer **endpoint** with the **IMPLEMENTATION-VERSION** attribute as specified in section 2.2.2.2. The additional messages MUST NOT be sent for **ICE keep-alive messages**. When a connectivity request or response is received, the fingerprint checks MUST use the fingerprint mechanism specified in [IETFDRAFT-ICENAT-19] section 7.1.1. If the fingerprint checks fail and the connectivity check request or response does not have the **IMPLEMENTATION-VERSION** attribute, the fingerprint checks SHOULD<10> be tried by using the CRC table shown earlier in this section. If the fingerprint checks succeed with the CRC table, the packet SHOULD<11> be considered a valid packet, and processed as such.

3.1.4.8.2.1 Forming the Candidate Pairs

After the **offer** and **answer** exchange of the **candidates** is finished, both **endpoints** have a set of local and **remote candidates**. The **local candidates** and remote candidates are paired together to form the **candidate pairs**. Local candidates and remote candidates with the same transport protocol and IP address family MUST be paired together to form candidate pairs. Local candidates and remote candidates with different transport protocols MUST NOT be paired together to form candidate pairs.

Each candidate pair MUST consist of four **transport addresses**: one for the **Real-Time Transport Protocol (RTP) component** for the local candidate, one for the RTP component for the remote candidate, one for the **Real-Time Transport Control Protocol (RTCP)** component for the local candidate, and one for the RTCP component for the remote candidate. For a candidate pair, the components of the local candidate MUST be paired with the corresponding components of the remote candidate to form a component pair. For example, the local candidate's RTP component transport address is paired with the remote candidate's RTP component transport address. **Transmission**

Control Protocol (TCP) candidate pairs MUST be formed as specified in [IETFDRAFT-ICETCP-07] section 4.2. Endpoints implementing this protocol MUST NOT generate more than 80 candidate pairs.<12>

3.1.4.8.2.2 Ordering the Candidate Pairs

The priorities for **candidate pairs** MUST be computed as specified in [IETFDRAFT-ICENAT-19] section 5.7.2. The candidate pairs MUST be ordered and pruned to form the **Check List** of candidate pairs, as specified in [IETFDRAFT-ICENAT-19] section 5.7.3.

3.1.4.8.2.3 Updating the Candidate Pair States

Each **candidate pair** state is updated as the **connectivity checks** progress. The candidate pair states and the transitions between the different states are specified in [IETFDRAFT-ICENAT-19] section 5.7.4.

3.1.4.8.2.4 Forming and Sending Binding Requests for Connectivity Checks

Connectivity checks are performed between the two **endpoints** by sending peer-to-peer **Simple Traversal of UDP through NAT (STUN)** binding request messages, as specified in [IETFDRAFT-ICENAT-19] section 5.8. The STUN binding request message MUST have the **USERNAME** and **MESSAGE-INTEGRITY** attributes and MUST use the STUN short-term credential mechanism. The **USERNAME** attribute MUST be formed as specified in [IETFDRAFT-ICENAT-19] section 7.1.1, which refers to the newer STUN message format. Make sure to use the older STUN message format specified in section 2.2.2, to perform the **MESSAGE-INTEGRITY** computation. Mandating the use of the **MESSAGE-INTEGRITY** attribute in STUN binding request messages serves to mitigate attacks on connectivity, as described in section 5.1.3. These two attributes MUST support the additional attributes specified in section 2.2 and MUST follow the usage specified in [IETFDRAFT-ICENAT-19] section 7.1.1. The connectivity checks MUST use the fingerprint mechanism as specified in [IETFDRAFT-ICENAT-19] section 7.1.1.

The STUN binding request message MUST have the **CANDIDATE-IDENTIFIER** attribute. The value of this attribute MUST be set to the **foundation** of the **local candidate** for which the request is being sent if the **candidate** is not a **peer-derived candidate**. If the local candidate is a peer-derived candidate, the value of **CANDIDATE-IDENTIFIER** MUST be set to the foundation of the peer-derived local candidate's **base**.

The connectivity checks are sent between **component** pairs based on the ordering of **candidate pairs** in the **Check List**, following the procedures specified in [IETFDRAFT-ICENAT-19] section 5.8. The processing of connectivity checks and the responses are specified in section 3.1.5.

3.1.4.8.2.5 Spacing the Connectivity Checks

To avoid flooding the network, the **connectivity checks** and their retries SHOULD be spaced as specified in [IETFDRAFT-ICENAT-19] section 5.8.

3.1.4.8.2.6 Terminating the Connectivity Checks

The **connectivity checks** phase MUST be terminated either when the **Connectivity Checks** timer is triggered or when the connectivity checks for all **candidate pairs** are complete. Connectivity checks for a candidate pair MUST be considered complete if the candidate pair is in either the "Succeeded" or the "Failed" state. At the end of the connectivity checks phase, if there are no candidate pairs in the **Valid List** on the **controlling agent**, the call MUST fail. On the controlling agent, the **endpoint** MUST begin performing **Regular Nomination**, as specified in [IETFDRAFT-ICENAT-19] section 8.1.1.1, for the candidate pair with the highest priority in the Valid List. If the nomination connectivity checks are successful, the **nominated** candidate pair MUST be selected for the final media flow. If the Regular Nomination connectivity checks fail, the call MUST fail. The controlling agent MUST respond to

connectivity checks until it gets the **answer** to its **final offer**. The **controlled agent** MUST continue to respond to connectivity checks until it gets the final offer from the controlling agent.

3.1.4.8.3 Media Flow

This section specifies the **candidate pair** that is used for media flow during processing, as designated by this protocol. Applications in regular mode can begin sending media after the initial exchange of **candidates** is finished. **Endpoints** that follow this protocol SHOULD be prepared to accept media on any of the **base transport addresses** of the published candidates. Any media sent at this stage MUST be sent using the **default candidate pair**. However, there is no guarantee that the media will reach the **peer** at this stage. During the **connectivity checks** phase, media SHOULD be switched to use the first candidate pair that has both of its constituent **component** pairs in the "Succeeded" state. After the final exchange of the candidates selected by the connectivity checks phase, media flow MUST be switched to use the best local and **remote candidates** exchanged. Applications in TCP-only mode MUST wait for connectivity checks to complete if they require data to be delivered reliably.

3.1.5 Message Processing Events and Sequencing Rules

3.1.5.1 Processing TURN Messages

The processing of **Traversal Using Relay NAT (TURN)** messages, response generation, and error handling is performed as specified in [MS-TURN] section 3.2.5 when communicating with a **TURN server**. The protocol also has support for communicating with a TURN server using [RFC5766].<13>

3.1.5.2 Processing STUN Messages

This protocol sends peer-to-peer **Simple Traversal of UDP through NAT (STUN)** messages between **endpoints** during the **connectivity checks** phase to select the **candidate pairs** for streaming media.

This section specifies the processing of STUN binding request messages by the two endpoints.

During the connectivity checks phase, whenever a connectivity check request is sent following the message format as specified in [IETFDRAFT-STUN-02], an additional connectivity check request or response SHOULD be sent following the message format as specified in [RFC5389]. The transmission of this additional connectivity check packet SHOULD be stopped on receiving a valid connectivity check request or response from the **peer** endpoint.

The first time a valid STUN message is received from the peer endpoint, the value of the **IMPLEMENTATION-VERSION** attribute MUST be read. A value greater than or equal to 0x00000003 implies that the peer endpoint supports [RFC5389] message formats. A value less than 0x00000003, implies that the peer endpoint supports only [IETFDRAFT-STUN-02] message formats. If the attribute does not exist, it implies that the peer endpoint supports [RFC5389] message formats only.

All subsequent connectivity check messages MUST follow the message format supported by the peer endpoint as detected based on the **IMPLEMENTATION-VERSION** attribute above.

3.1.5.2.1 Processing the STUN Binding Request

The **Simple Traversal of UDP through NAT (STUN)** binding request messages might be received before the **remote candidates** are received from the **peer endpoint** in the **offer** or **answer**. The endpoint MUST validate the request. If the request is invalid, the endpoint SHOULD send a binding error response for the STUN binding request message, as specified in section 3.1.5.2.2. If the request is valid, the endpoint MUST follow the procedures specified in [IETFDRAFT-ICENAT-19] section 7.2 for processing the STUN binding request.

3.1.5.2.2 Validating the STUN Binding Request

The validation procedures for **Simple Traversal of UDP through NAT (STUN)** binding request messages as specified in [IETFDRAFT-STUN-02] section 8 differ from the procedures described in this section. **Endpoints** that follow this protocol MUST follow the procedures in this section to validate the STUN binding request messages that are received for **connectivity checks**.

If a STUN binding request message is received without a **USERNAME** attribute, the STUN binding request message MUST be discarded. **ICE keep-alive messages** are discarded if they do not have the **USERNAME** attribute. If the **USERNAME** attribute is not valid, the message MUST be discarded. A **USERNAME** attribute is considered valid if it consists of two values separated by a colon and the first value equals the **username** fragment generated by the endpoint in the **offer**. If the received STUN binding request message does not have the **fingerprint** attribute, the message MUST be discarded. If the STUN binding request message does not have the **MESSAGE-INTEGRITY** attribute, the endpoint MUST send a binding error response with error code 401 (Unauthorized), as specified in [IETFDRAFT-STUN-02] section 8. If the **MESSAGE-INTEGRITY** attribute exists, the endpoint MUST use the STUN short-term credential mechanism, by using the password that was sent to the **peer** to compute the message integrity, and verify against the message integrity value in the request. If the message integrity check fails, the endpoint MUST send a binding error response with error code 431 (Integrity Check Failure), as specified in [IETFDRAFT-STUN-02] section 8. Generated binding error responses MUST have a **USERNAME** attribute set to the value of the **USERNAME** attribute received in the STUN binding request message.

3.1.5.2.3 Sending the STUN Binding Response

If the request is valid, the **endpoint** MUST send a **Simple Traversal of UDP through NAT (STUN)** binding response message, as specified in [IETFDRAFT-STUN-02] section 8, with a subset of its attributes. The STUN binding response message MUST implement only the following attributes:

- **XOR-MAPPED-ADDRESS**
- **USERNAME**
- **MESSAGE-INTEGRITY**
- **IMPLEMENTATION-VERSION**

The format of the **XOR-MAPPED-ADDRESS** attribute MUST be as specified in [IETFDRAFT-STUN-02] section 8.1. The **XOR-MAPPED-ADDRESS** attribute MUST have a value of 0x0020. The **X-PORT** and **X-ADDRESS** fields MUST be computed as specified in [IETFDRAFT-STUN-02] section 8.1 for the IP address and port from which the STUN binding request message was received. The **USERNAME** attribute MUST be formed as specified in [IETFDRAFT-ICENAT-19] section 7.1.1, which refers to the newer STUN message format. Make sure to use the STUN message format specified in section 2.2.2, to perform the **MESSAGE-INTEGRITY** computation.

3.1.5.3 STUN Binding Response

This section specifies the way an **endpoint** processes **Simple Traversal of UDP through NAT (STUN)** binding response messages. The processing consists of two tasks. The first task is the validation of the STUN binding response message. The second task is the **connectivity check** processing, which includes updating the state of the **component** pairs and discovering **peer-derived candidates**. The procedures for processing STUN binding responses MUST be performed as specified in [IETFDRAFT-ICENAT-19] section 7.1.2.

3.1.5.3.1 Validating the STUN Binding Response

If a **Simple Traversal of UDP through NAT (STUN)** binding response message is received before the **peer's candidates** are received through the **offer** exchange, it MUST be discarded. If a STUN binding response message is received without a **USERNAME** attribute, it MUST be discarded.<14> If the **component** pair is in a failed state, the STUN binding response message MUST be discarded. If

the received STUN binding response message does not have the **fingerprint** attribute, the message MUST be discarded.

The password received from the peer **endpoint** is used to compute the message integrity. The computed message integrity value MUST be verified against the **MESSAGE-INTEGRITY** attribute value in the message. If the message integrity check fails, the STUN binding response message MUST be discarded. If the message does not have the **XOR-MAPPED-ADDRESS** attribute, the STUN binding response message MUST be discarded. If the IP address in **XOR-MAPPED-ADDRESS** is null ("0.0.0.0"), "Broadcast", or "Multicast", the STUN binding response message MUST be discarded.

3.1.5.3.2 Processing the STUN Binding Response

The procedures for processing the **Simple Traversal of UDP through NAT (STUN)** binding response MUST be performed as specified in [IETFDRAFT-ICENAT-19] section 7.1.2.

3.1.5.3.3 STUN Binding Error Response

The error response message MUST be validated in the same way as **Simple Traversal of UDP through NAT (STUN)** binding response messages. The validation procedure is specified in section 3.1.5.3.2.

If the **component** for which the error response is received is already in the "Succeeded" state, the error response message MUST be discarded. If the error code in the error response message is 401, 430, 431, 432, or 500, **connectivity checks** for the **transport address** SHOULD be retried. If any other error code is received in the binding error response message, the component pair MUST be set to a "Failed" state.

3.1.6 Timer Events

3.1.6.1 Candidates Gathering Phase Timer

The firing of the **Candidates Gathering Phase** timer signals the end of the **candidates** gathering phase. The **endpoint** MUST exchange the gathered **local candidates** with its **peer**.

3.1.6.2 Connectivity Checks Phase Timer

After a **Simple Traversal of UDP through NAT (STUN)** binding request message and response are received from the **peer**, the **Connectivity Checks Phase** timer MUST be reset to 5 seconds. The firing of this timer signals the end of the **connectivity checks** phase. When this timer fires, the **controlling agent** MUST pick the best **candidate pair** selected by the connectivity checks and send it to the **controlled agent**. If no candidate pair is validated by the connectivity checks when the timer fires on the controlling agent, the call MUST fail. Further connectivity check attempts MUST NOT be made after this timer fires. When this timer fires on the controlled agent, it MUST stop its connectivity checks.

3.1.6.3 ICE keep-alive Timer

The **ICE keep-alive messages** are sent from the **local transport address** to the remote **transport address** in the **component** pair. ICE keep-alive messages MUST be sent even if the **peer endpoint** does not implement ICE for the **Real-Time Transport Protocol (RTP)** component pair that is associated with the **candidate pair** that is used for media flow. ICE keep-alive messages MUST be **Simple Traversal of UDP through NAT (STUN)** binding request messages, as specified in section 2.2.3.

During the **connectivity checks** phase, the **ICE keep-alive** timer SHOULD fire<15> for validated component pairs and for component pairs whose **local candidates** are **Relayed Candidates**, if no

connectivity check packets or ICE keep-alive messages have been sent for the component pair for the duration of the timer value. When the **ICE keep-alive** timer fires, an ICE keep-alive message SHOULD be sent for the component pair.

In addition to the keep-alive messages during the connectivity checks, for the **candidate** that is being used for media flow, the **ICE keep-alive** timer MUST fire when there has been no flow of media or ICE keep-alive messages for the duration of the timer. When the **ICE keep-alive** timer fires, an ICE keep-alive message MUST be sent for the RTP component pair that is associated with the media flow candidate pair. ICE keep-alive messages SHOULD NOT be sent for a **Real-Time Transport Control Protocol (RTCP)** component because the flow of **RTCP packets** is sufficient to keep the **NAT bindings** and **Traversal Using Relay NAT (TURN)** allocations active.

3.1.6.4 USE-CANDIDATE Checks Timer

If the USE-CANDIDATE checks timer expires and if the nomination checks have not been completed then the call MUST be failed.

3.1.6.5 Consent Freshness Timer

After ICE connectivity checks has successfully established a media path, the consent freshness timer MUST<16> be reset to 30 seconds. Consent request messages are sent out periodically on the **nominated** path. The interval between consent request messages SHOULD be 5 seconds.

Consent request messages MUST be valid STUN binding request messages formed as described in section 3.1.4.8.2.4. However, it SHOULD NOT include the **CANDIDATE-IDENTIFIER** attribute. It MUST follow the [RFC5389] message format. The transaction ID, as defined in [RFC5389] section 6, MUST be re-generated for each consent request message sent.

On receiving a consent request message that is sent along the nominated path, the **endpoint** MUST validate the message as described in section 3.1.5.2.2. However, it SHOULD NOT send any STUN binding error response messages. Instead, the message MUST be discarded if **MESSAGE-INTEGRITY** validation fails. Once the message has been validated, a consent response message MUST be sent. This MUST be a STUN binding response as described in section 3.1.5.2.3 and MUST follow the [RFC5389] message format.

On receiving a consent response message where the transaction ID corresponds to the transaction ID of the last sent consent request message, the **FINGERPRINT** and **MESSAGE-INTEGRITY** attributes MUST be validated according to [RFC5389]. If the message is valid, the consent freshness timer MUST be reset to 30 seconds.

After the consent freshness timer expires, the endpoint SHOULD terminate the media session. All consent response messages received after the timer expires MUST be ignored.

3.1.7 Other Local Events

None.

4 Protocol Examples

This protocol example illustrates the establishment of a media session between two endpoints based on the sample topology that is shown in the following figure.

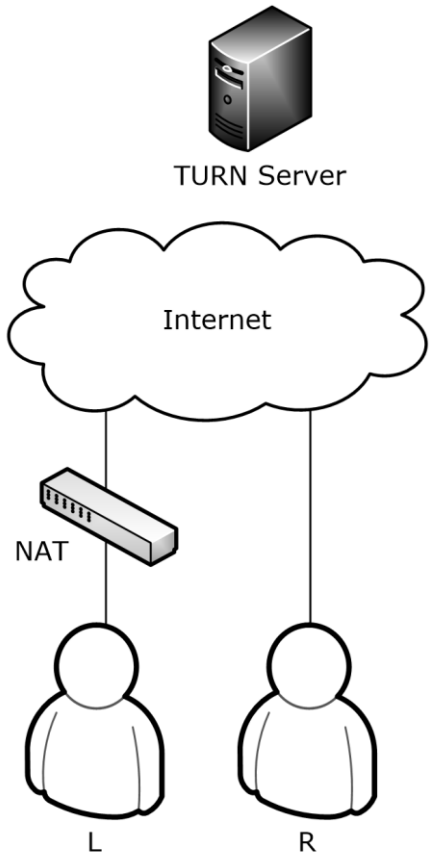


Figure 3: ICE implementations

The figure shows Endpoint L and Endpoint R using ICE. Both **agents** are **full** ICE implementations and use **Regular Nominations** for selecting the **candidates** to be used for media flow. Endpoint L is behind a **NAT** device in a private address space (192.168.2.1) with the public edge of the NAT device at 10.107.0.71, and Agent R is on the public Internet at 10.104.0.68. Both **endpoints** are configured with the same **User Datagram Protocol (UDP) TURN server** that is listening on IP address 10.101.0.57and port 3478.

The **transport address** follows a similar naming convention to that in the sample described in [IETFDRAFT-ICENAT-19] section 17.

Transport addresses are referred to by using the mnemonic names with the format *entity-type-seqno*, where *entity* refers to the entity whose IP address the transport address is on and is either "L", "R", "NAT", or "TURN". The *type* is either "PUB" for transport addresses that are publicly reachable on the Internet or "PRIV" for transport addresses that are not reachable from the Internet. The *seqno* is a number that is different for transport addresses of the same type on an entity. The TURN server has the transport address TURN-PUB-1 (10.101.0.57 and port 3478).

For the call flow:

- "S=" refers to the source transport address.

- "D=" refers to the destination transport address.
- "MA=" refers to the mapped address in the **Simple Traversal of UDP through NAT (STUN)** binding response.
- "RA=" refers to the reflexive address.
- "TA=" refers to the relay transport address.

For clarity, the example does not show the **Traversal Using Relay NAT (TURN) authentication** mechanisms and the **Real-Time Transport Control Protocol (RTCP) component**.

The example focuses on the **Real-Time Transport Protocol (RTP)** component for establishing a media session between Endpoint L and Endpoint R. Endpoint L initiates the media session and becomes the **controlling agent** because Endpoint L is a full ICE implementation. Endpoint L gathers its UDP **Host Candidate** by binding to its local interface and then gathers UDP **Relayed Candidates** and UDP **Server Reflexive Candidates** from the configured TURN server. Because no **Transmission Control Protocol (TCP)** TURN servers are configured, Endpoint L creates an active TCP TCP-ACT Server Reflexive Candidate based on the UDP Server Reflexive Candidate. After gathering the candidates, Endpoint L sends the **INVITE** to Endpoint R. A sample INVITE **Session Description Protocol (SDP)** for Endpoint L's topology is as follows:

```
v=0
o=- 0 0 IN IP4 10.101.0.57
s=session
c=IN IP4 10.101.0.57
b=CT:99980
t=0 0
m=audio 52732 RTP/AVP 114 111 112 115 116 4 8 0 97 13 118 101
a=ice-ufrag:qkEP
a=ice-pwd:ed6f9GuHjLcoCN6sC/Eh7fVl
a=candidate:1 1 UDP 2130706431 192.168.2.1 50005 typ host
a=candidate:2 1 UDP 16648703 10.101.0.57 52732 typ relay raddr 10.107.0.71 rport 50033
a=candidate:3 1 UDP 1694234623 10.107.0.71 50033 typ srflx raddr 192.168.2.1 rport 50033
a=candidate:4 1 TCP-ACT 1684797951 10.107.0.71 50033 typ srflx raddr 192.168.2.1 rport 50033
a=rtpmap:114 x-msrta/16000
```

The following diagrams illustrate the ICE request and response sequence.

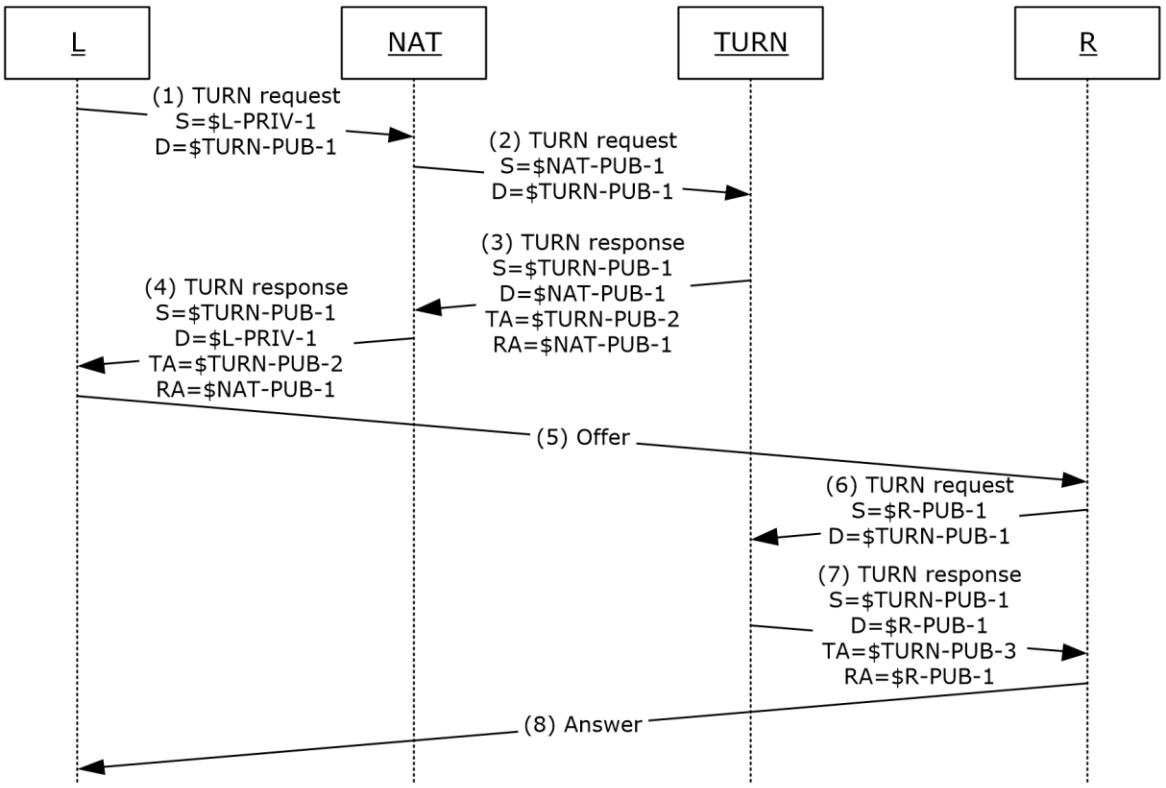


Figure 4: ICE request and response sequence

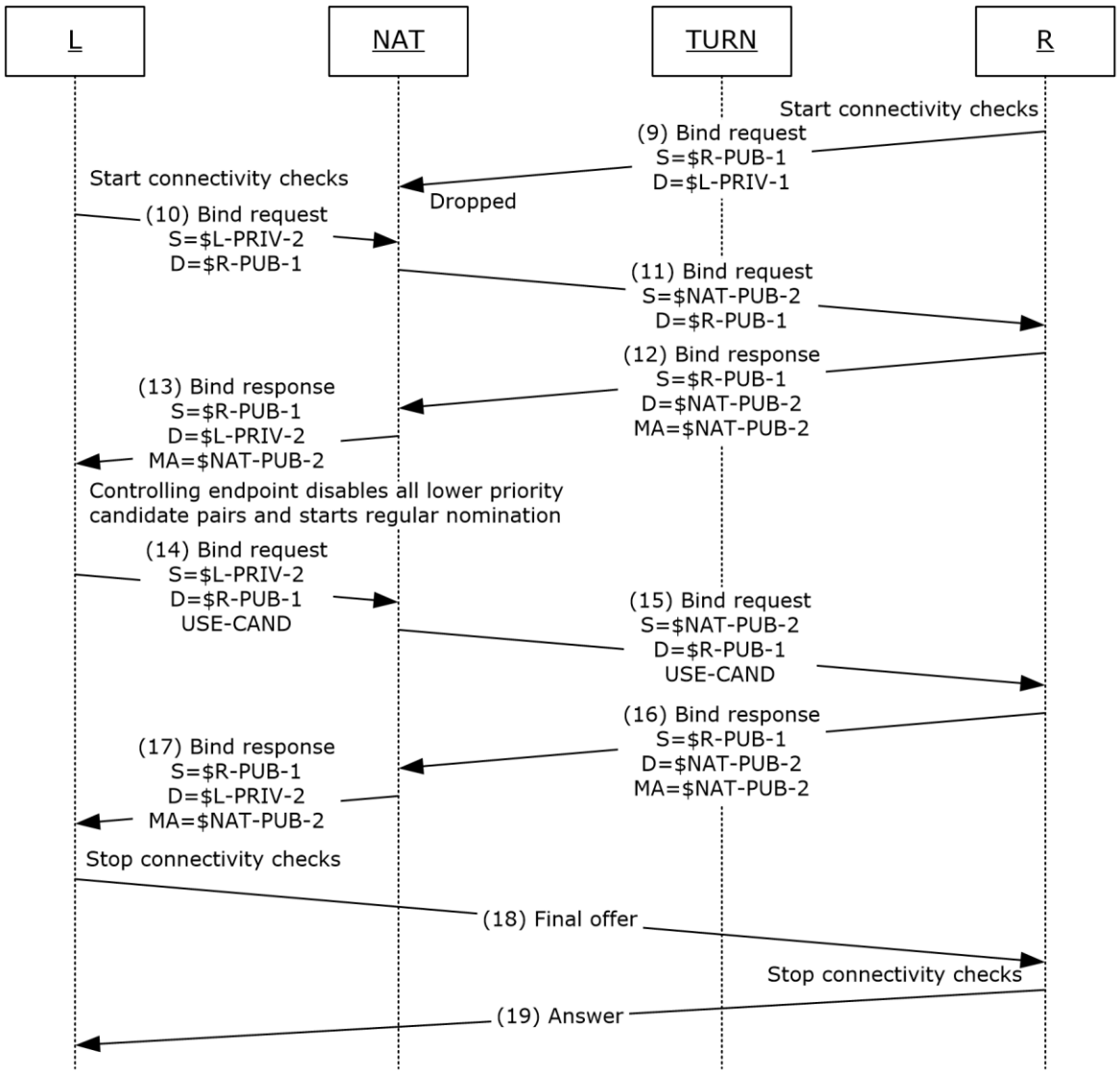


Figure 5: ICE request and response sequence (continued)

Endpoint R, upon receiving the **offer**, gathers its candidates. It gathers its UDP Host Candidate by binding to its local interface and then gathers UDP Relayed Candidates from the configured TURN server. Endpoint R is not behind a NAT device so UDP Server Reflexive Candidates are created. Because no TCP TURN servers are configured, Endpoint R creates a TCP-ACT Server Reflexive Candidate based on the UDP Host Candidate. Endpoint R sends its candidates to Endpoint L in the **answer**. Endpoint R pairs its **local candidates** with Endpoint L's **remote candidates** and starts **connectivity checks**. A sample answer SDP for Endpoint R's topology is as follows:

```
v=0
o=- 0 0 IN IP4 10.101.0.57
s=session
c=IN IP4 10.101.0.57
b=CT:99980
t=0 0
m=audio 52714 RTP/AVP 114 111 112 115 116 4 8 0 97 13 118 101
a=ice-ufrag:qkEP
a=ice-pwd:ed6f9GuHjLcoCN6sC/Eh7fVl
a=candidate:1 1 UDP 2130706431 10.104.0.68 50025 typ host
```

```
a=candidate:2 1 UDP 16648703 10.101.0.57 52714 typ relay raddr 10.104.0.68 rport 50036
a=candidate:3 1 TCP-ACT 1684797951 10.104.0.68 50025 typ srflx raddr 10.104.0.68 rport 50025
a=rtpmap:114 x-msrta/16000
```

Endpoint L, upon receiving the answer from Endpoint R, pairs its local candidates with the candidates received in the answer and starts connectivity checks. Both endpoints perform connectivity checks with the highest priority **candidate pairs**.

The preceding sequence diagram shows that Endpoint R sends a STUN binding request from R-PUB-1 to L-PRIV-2, which does not reach L-PRIV-2 because it is not directly reachable from R-PUB-1. At this point, Endpoint L sends a STUN binding request from L-PRIV-2 to R-PUB-1. This request goes through the NAT device and Endpoint R eventually receives the packet at R-PUB-1 with the source as NAT-PUB-2. Agent R sends a STUN binding response with the mapped address set to NAT-PUB-2. Endpoint L eventually gets the packet from the NAT device and discovers a new **peer-derived candidate**, because the mapped address is different from the address the STUN binding request sent. The endpoint validates this candidate pair and disables all lower priority candidate pairs. Because this is the highest priority candidate pair, Endpoint L nominates this candidate pair and sends a STUN binding request to R-PUB-1 with the **USE-CANDIDATE** flag set. Endpoint R, upon getting the request with the **USE-CANDIDATE** flag, responds with a STUN binding response. Upon receiving the response, Endpoint L stops its connectivity checks because it has found the candidate pair that has to be used for media flow.

Endpoint L sends the **final offer** to Endpoint R, with the final local and remote candidate to be used for media flow. A sample final offer is as follows:

```
v=0
o=- 0 0 IN IP4 10.107.0.71
s=session
c=IN IP4 10.107.0.71
b=CT:99980
t=0 0
m=audio 50005 RTP/SAVP 114 111 112 115 116 4 8 0 97 13 118 101
a=ice-ufrag:32sD
a=ice-pwd:YF9/OwRcN/pXUglBv1c+5QMu
a=candidate:7 1 UDP 1862270719 10.107.0.71 50005 typ prflx raddr 192.168.2.4 rport 50005
a=remote-candidates:1 10.104.0.68 50025
a=rtpmap:114 x-msrta/16000
```

Endpoint R, upon receiving the final offer, stops its connectivity checks and sends its answer to the final offer:

```
v=0
o=- 0 0 IN IP4 10.104.0.68
s=session
c=IN IP4 10.104.0.68
b=CT:99980
t=0 0
m=audio 50025 RTP/SAVP 114 111 112 115 116 4 8 0 97 13 118 101
a=ice-ufrag:32sD
a=ice-pwd:YF9/OwRcN/pXUglBv1c+5QMu
a=candidate:7 1 UDP 1862270719 10.104.0.68 50025 typ host
a=remote-candidates:1 10.107.0.71 50005
a=rtpmap:114 x-msrta/16000
```

With the receipt of the final answer, the connectivity checks phase ends and both ends stream media using the final candidates selected by the connectivity checks.

5 Security

5.1 Security Considerations for Implementers

This protocol has similar security concerns as those described in [IETFDRAFT-ICENAT-19] section 18. Additional considerations and mitigations pertaining to this protocol are listed in this section.

5.1.1 Attacks on Address Gathering

The security considerations for using the protocol described in [MS-TURN] for gathering **STUN candidates** and **TURN candidates** are described in [MS-TURN] section 5.

5.1.2 Attacks on Connectivity Checks

An attacker might attempt to sniff the signaled **candidates** and passwords to maliciously obtain control of the call and related media. This protocol relies on the existence of a secure channel to exchange candidates. A malicious user might attempt to attack the **Simple Traversal of UDP through NAT (STUN) connectivity checks**, either to maliciously gain control of the call and related media to a different **endpoint** or to cause a failure of the connectivity checks. The malicious user can potentially inject connectivity check packets to fool an endpoint into considering a valid **candidate pair** invalid or vice versa. Alternatively, the malicious user can cause the endpoints to discover incorrect **peer-derived candidates**. These attacks are mitigated by this protocol by mandating the **MESSAGE-INTEGRITY** attribute in the STUN connectivity checks and responses.

5.1.3 Voice Amplification Attack

A malicious user can include the target address of the denial of service attack as the **default candidate** in its **offer** and send the offer to multiple **endpoints**. This action can potentially result in each endpoint that received the offer attempting to send media to the target of the denial of service attack. This attack can be mitigated by using this protocol in conjunction with a secure signaling layer for offer exchange that is associated with targeted **candidates** and associated credentials.

5.1.4 STUN Amplification Attack

The **Simple Traversal of UDP through NAT (STUN)** amplification attack is similar to the voice amplification attack. Instead of media flow, the STUN **connectivity checks** are directed to the target of the denial of service attack. The malicious user proceeds by generating an **offer** with a large number of **candidates** for the denial of service target. The **peer endpoint**, after receiving the offers, performs connectivity checks with all the candidates specified in the offer. This malicious activity can generate a significant volume of data flow with STUN connectivity checks. This malicious activity cannot be completely prevented by this protocol, but the protocol can mitigate this type of malicious activity to a certain extent by limiting the total number of candidates that are sent in an offer and response to 20 candidates and 40 **candidate pairs**. This protocol mitigates the similar attack of generating multiple **provisional answers** to an offer by limiting the number of provisional answers supported. In addition, this protocol relies on a secure signaling layer for offer exchanges of candidates and associated user names and passwords.

5.2 Index of Security Parameters

None.

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include updates to those products.

- Microsoft Office Communicator 2007
- Microsoft Office Communications Server 2007
- Microsoft Office Communications Server 2007 R2
- Microsoft Office Communicator 2007 R2
- Microsoft Lync 2010
- Microsoft Lync Server 2010
- Microsoft Lync Client 2013/Skype for Business
- Microsoft Lync Server 2013
- Microsoft Skype for Business 2016
- Microsoft Skype for Business Server 2015
- Windows 10 v1511 operating system
- Windows Server 2016 operating system
- Windows Server operating system

Exceptions, if any, are noted in this section. If an update version, service pack or Knowledge Base (KB) number appears with a product name, the behavior changed in that update. The new behavior also applies to subsequent updates unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms "SHOULD" or "SHOULD NOT" implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term "MAY" implies that the product does not follow the prescription.

<1> Section 2.1: Office Communicator 2007 R2, Office Communications Server 2007 R2, Lync 2010, Lync Server 2010: **IPv6** is not supported.

<2> Section 2.2.2: Skype for Business 2016, Skype for Business Server 2015, Windows 10 v1511, Windows 10 v1511 Enterprise operating system, Windows Server 2016: [RFC5389] is applicable only to these versions and higher.

<3> Section 2.2.2.2: Office Communicator 2007, Office Communications Server 2007: The **IMPLEMENTATION-VERSION** attribute is not supported.

<4> Section 2.2.2.2: Office Communicator 2007 R2, Office Communications Server 2007 R2: This value is set to 0x00000001. Lync 2010, Lync Server 2010, Lync Client 2013/Skype for Business, Lync Server 2013: This value is set to 0x00000002.

<5> Section 3.1.4.4.1: Office Communicator 2007, Office Communications Server 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2, Lync 2010, Lync Server 2010, Lync Client 2013/Skype for Business, Lync Server 2013, Skype for Business 2016, Skype for Business Server 2015, Windows 10 v1511: ICE **Lite** handling described in this section applies only to these versions.

<6> Section 3.1.4.8.1: Office Communicator 2007 R2, Office Communications Server 2007 R2, Lync 2010, Lync Server 2010: IPv6 is not supported.

<7> Section 3.1.4.8.1: Office Communicator 2007 R2, Office Communications Server 2007 R2, Lync 2010, Lync Server 2010: A maximum of 20 candidates is supported.

<8> Section 3.1.4.8.1.1: Office Communicator 2007 R2, Office Communications Server 2007 R2, Lync 2010, Lync Server 2010: IPv6 is not supported.

<9> Section 3.1.4.8.2: Office Communications Server 2007 R2, Office Communicator 2007 R2: This behavior is not supported. Lync Server 2013, Lync Client 2013/Skype for Business: This behavior is not supported for IPv6 candidate pairs.

<10> Section 3.1.4.8.2: Office Communications Server 2007 R2, Office Communicator 2007 R2: This behavior is not supported.

<11> Section 3.1.4.8.2: Office Communications Server 2007 R2, Office Communicator 2007 R2: This behavior is not supported.

<12> Section 3.1.4.8.2.1: Office Communicator 2007 R2, Office Communications Server 2007 R2, Lync 2010, Lync Server 2010: A maximum of 40 candidate pairs is supported.

<13> Section 3.1.5.1: Windows 10 v1511, Windows 10 v1511 Enterprise, Windows Server 2016: Applicable only to these versions and higher.

<14> Section 3.1.5.3.1: Office Communicator 2007, Office Communications Server 2007, Office Communications Server 2007 R2, Office Communicator 2007 R2, Lync 2010, Lync Server 2010, Lync Client 2013/Skype for Business, Lync Server 2013: Applicable only to these versions.

<15> Section 3.1.6.3: Office Communicator 2007 R2, Office Communications Server 2007 R2: This behavior is not supported.

<16> Section 3.1.6.5: Skype for Business 2016, Skype for Business Server 2015, Windows 10 v1511, Windows 10 v1511 Enterprise, Windows Server 2016: Applicable only to these versions and higher.

7 Change Tracking

This section identifies changes that were made to this document since the last release. Changes are classified as Major, Minor, or None.

The revision class **Major** means that the technical content in the document was significantly revised. Major changes affect protocol interoperability or implementation. Examples of major changes are:

- A document revision that incorporates changes to interoperability requirements.
- A document revision that captures changes to protocol functionality.

The revision class **Minor** means that the meaning of the technical content was clarified. Minor changes do not affect protocol interoperability or implementation. Examples of minor changes are updates to clarify ambiguity at the sentence, paragraph, or table level.

The revision class **None** means that no new technical changes were introduced. Minor editorial and formatting changes may have been made, but the relevant technical content is identical to the last released version.

The changes made to this document are listed in the following table. For more information, please contact dochelp@microsoft.com.

Section	Description	Revision class
6 Appendix A: Product Behavior	Added product to the applicable products list.	Major

8 Index

V