Stream:	Internet Engineering Task Force (IETF)
RFC:	8838
Category:	Standards Track
Published:	January 2021
ISSN:	2070-1721
Authors:	E. Ivov	J. Uberti	P. Saint-Andre
---	---	---	---
	8x8 / Jitsi	*Google*	*Mozilla*

RFC 8838
Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol

Abstract

This document describes "Trickle ICE", an extension to the Interactive Connectivity Establishment (ICE) protocol that enables ICE agents to begin connectivity checks while they are still gathering candidates, by incrementally exchanging candidates over time instead of all at once. This method can considerably accelerate the process of establishing a communication session.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8838.

Copyright Notice

Table of Contents

Authors' Addresses

1. Introduction

The Interactive Connectivity Establishment (ICE) protocol [RFC8445] describes how an ICE agent gathers candidates, exchanges candidates with a peer ICE agent, and creates candidate pairs. Once the pairs have been gathered, the ICE agent will perform connectivity checks and eventually nominate and select pairs that will be used for sending and receiving data within a communication session.

Following the procedures in [RFC8445] can lead to somewhat lengthy establishment times for communication sessions, because candidate gathering often involves querying Session Traversal Utilities for NAT (STUN) servers [RFC5389] and allocating relayed candidates on Traversal Using Relay NAT (TURN) servers [RFC5766]. Although many ICE procedures can be completed in parallel, the pacing requirements from [RFC8445] still need to be followed.

This document defines "Trickle ICE", a supplementary mode of ICE operation in which candidates can be exchanged incrementally as soon as they become available (and simultaneously with the gathering of other candidates). Connectivity checks can also start as soon as candidate pairs have been created. Because Trickle ICE enables candidate gathering and connectivity checks to be done in parallel, the method can considerably accelerate the process of establishing a communication session.

This document also defines how to discover support for Trickle ICE, how the procedures in [RFC8445] are modified or supplemented when using Trickle ICE, and how a Trickle ICE agent can interoperate with an ICE agent compliant to [RFC8445].

This document does not define any protocol-specific usage of Trickle ICE. Instead, protocol-specific details for Trickle ICE are defined in separate usage documents. Examples of such documents are [RFC8840] (which defines usage with the Session Initiation Protocol (SIP) [RFC3261] and the Session Description Protocol (SDP) [RFC4566]) and [XEP-0176] (which defines usage with the Extensible Messaging and Presence Protocol (XMPP) [RFC6120]). However, some of the examples in the document use SDP and the Offer/Answer model [RFC3264] to explain the underlying concepts.

The following diagram illustrates a successful Trickle ICE exchange with a using protocol that follows the Offer/Answer model:

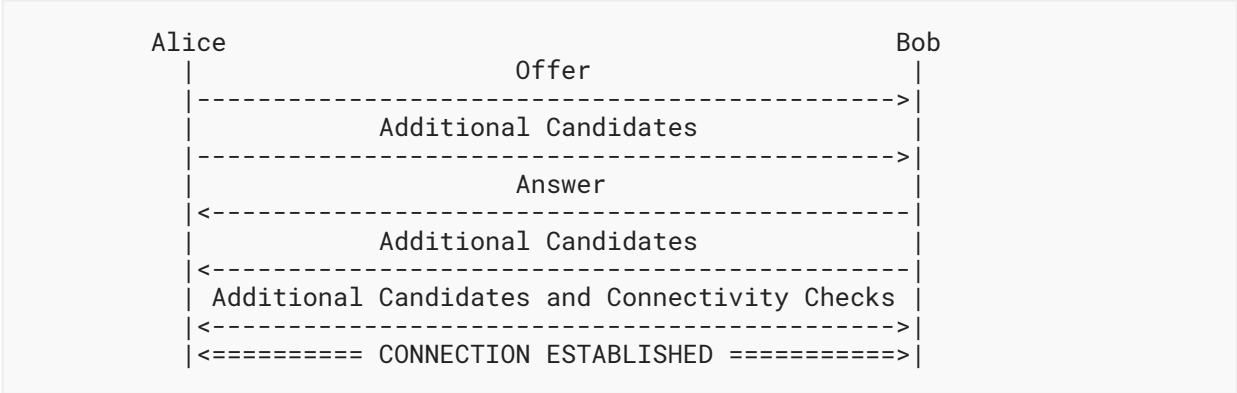

Figure 1: Flow

The main body of this document is structured to describe the behavior of Trickle ICE agents in roughly the order of operations and interactions during an ICE session:

1. Determining support for Trickle ICE
2. Generating the initial ICE description
3. Handling the initial ICE description and generating the initial ICE response
4. Handling the initial ICE response
5. Forming checklists, pruning candidates, performing connectivity checks, etc.
6. Gathering and conveying candidates after the initial ICE description and response
7. Handling inbound trickled candidates
8. Generating and handling the end-of-candidates indication
9. Handling ICE restarts

There is quite a bit of operational experience with the technique behind Trickle ICE, going back as far as 2005 (when the XMPP Jingle extension defined a "dribble mode" as specified in [XEP-0176]); this document incorporates feedback from those who have implemented and deployed the technique over the years.

2. Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification makes use of all terminology defined for Interactive Connectivity Establishment in [RFC8445]. In addition, it defines the following terms:

Empty Checklist: A checklist that initially does not contain any candidate pairs because they will be incrementally added as they are trickled. (This scenario does not arise with a regular ICE agent, because all candidate pairs are known when the agent creates the checklist set.)

Full Trickle: The typical mode of operation for Trickle ICE agents, in which the initial ICE description can include any number of candidates (even zero candidates) and does not need to include a full generation of candidates as in half trickle.

Generation: All of the candidates conveyed within an ICE session (correlated with a particular Username Fragment and Password combination).

Half Trickle: A Trickle ICE mode of operation in which the initiator gathers a full generation of candidates strictly before creating and conveying the initial ICE description. Once conveyed, this candidate information can be processed by regular ICE agents, which do not require support for Trickle ICE. It also allows Trickle-ICE-capable responders to still gather candidates and perform connectivity checks in a non-blocking way, thus providing roughly "half" the advantages of Trickle ICE. The half-trickle mechanism is mostly meant for use when the responder's support for Trickle ICE cannot be confirmed prior to conveying the initial ICE description.

ICE Description: Any attributes related to the ICE session (other than candidates) required to configure an ICE agent. These include but are not limited to the Username Fragment, the Password, and other attributes.

Trickled Candidates: Candidates that a Trickle ICE agent conveys after conveying or responding to the initial ICE description, but within the same ICE session. Trickled candidates can be conveyed in parallel with candidate gathering and connectivity checks.

Trickling: The act of incrementally conveying trickled candidates.

3. Determining Support for Trickle ICE

To fully support Trickle ICE, using protocols **SHOULD** incorporate one of the following mechanisms so that implementations can determine whether Trickle ICE is supported:

1. Provide a capabilities discovery method so that agents can verify support of Trickle ICE prior to initiating a session (XMPP's Service Discovery [XEP-0030] is one such mechanism).
2. Make support for Trickle ICE mandatory so that user agents can assume support.

If a using protocol does not provide a method of determining ahead of time whether Trickle ICE is supported, agents can make use of the half-trickle procedure described in Section 16.

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

Even if a using protocol does not include a capabilities discovery method, a user agent can provide an indication within the ICE description that it supports Trickle ICE by communicating an ICE option of 'trickle'. This token **MUST** be provided either at the session level or, if at the data stream level, for every data stream (an agent **MUST NOT** specify Trickle ICE support for some data

streams but not others). Note: The encoding of the 'trickle' ICE option, and the message(s) used to carry it to the peer, are protocol specific; for instance, the encoding for SDP [RFC4566] is defined in [RFC8840].

Dedicated discovery semantics and half trickle are needed only prior to initiation of an ICE session. After an ICE session is established and Trickle ICE support is confirmed for both parties, either agent can use full trickle for subsequent exchanges (see also Section 15).

4. Generating the Initial ICE Description

An ICE agent can start gathering candidates as soon as it has an indication that communication is imminent (e.g., a user-interface cue or an explicit request to initiate a communication session). Unlike in regular ICE, in Trickle ICE implementations do not need to gather candidates in a blocking manner. Therefore, unless half trickle is being used, the user experience is improved if the initiating agent generates and transmits its initial ICE description as early as possible (thus enabling the remote party to start gathering and trickling candidates).

An initiator **MAY** include any mix of candidates when conveying the initial ICE description. This includes the possibility of conveying all the candidates the initiator plans to use (as in half trickle), conveying only a publicly reachable IP address (e.g., a candidate at a data relay that is known to not be behind a firewall), or conveying no candidates at all (in which case the initiator can obtain the responder's initial candidate list sooner, and the responder can begin candidate gathering more quickly).

For candidates included in the initial ICE description, the methods for calculating priorities and foundations, determining redundancy of candidates, and the like work just as in regular ICE [RFC8445].

5. Handling the Initial ICE Description and Generating the Initial ICE Response

When a responder receives the initial ICE description, it will first check if the ICE description or initiator indicates support for Trickle ICE as explained in Section 3. If not, the responder **MUST** process the initial ICE description according to regular ICE procedures [RFC8445] (or, if no ICE support is detected at all, according to relevant processing rules for the using protocol, such as Offer/Answer processing rules [RFC3264]). However, if support for Trickle ICE is confirmed, a responder will automatically assume support for regular ICE as well.

If the initial ICE description indicates support for Trickle ICE, the responder will determine its role and start gathering and prioritizing candidates; while doing so, it will also respond by conveying an initial ICE response, so that both the initiator and the responder can form checklists and begin connectivity checks.

A responder can respond to the initial ICE description at any point while gathering candidates. The initial ICE response **MAY** contain any set of candidates, including all candidates or no candidates. (The benefit of including no candidates is to convey the initial ICE response as quickly as possible, so that both parties can consider the ICE session to be under active negotiation as soon as possible.)

As noted in Section 3, in using protocols that use SDP, the initial ICE response can indicate support for Trickle ICE by including a token of 'trickle' in the ice-options attribute.

6. Handling the Initial ICE Response

When processing the initial ICE response, the initiator follows regular ICE procedures to determine its role, after which it forms checklists (Section 7) and performs connectivity checks (Section 8).

7. Forming Checklists

According to regular ICE procedures [RFC8445], in order for candidate pairing to be possible and for redundant candidates to be pruned, the candidates would need to be provided in the initial ICE description and initial ICE response. By contrast, under Trickle ICE, checklists can be empty until candidates are conveyed or received. Therefore, a Trickle ICE agent handles checklist formation and candidate pairing in a slightly different way than a regular ICE agent: the agent still forms the checklists, but it populates a given checklist only after it actually has candidate pairs for that checklist. Every checklist is initially placed in the Running state, even if the checklist is empty (this is consistent with Section 6.1.2.1 of [RFC8445]).

8. Performing Connectivity Checks

As specified in [RFC8445], whenever timer Ta fires, only checklists in the Running state will be picked when scheduling connectivity checks for candidate pairs. Therefore, a Trickle ICE agent **MUST** keep each checklist in the Running state as long as it expects candidate pairs to be incrementally added to the checklist. After that, the checklist state is set according to the procedures in [RFC8445].

Whenever timer Ta fires and an empty checklist is picked, no action is performed for the list. Without waiting for timer Ta to expire again, the agent selects the next checklist in the Running state, in accordance with Section 6.1.4.2 of [RFC8445].

Section 7.2.5.4 of [RFC8445] requires that agents update checklists and timer states upon completing a connectivity check transaction. During such an update, regular ICE agents would set the state of a checklist to Failed if both of the following two conditions are satisfied:

- all of the pairs in the checklist are in either the Failed state or the Succeeded state; and
- there is not a pair in the valid list for each component of the data stream.

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

- all candidate gathering has completed, and the agent is not expecting to discover any new local candidates; and
- the remote agent has conveyed an end-of-candidates indication for that checklist as described in Section 13.

9. Gathering and Conveying Newly Gathered Local Candidates

After Trickle ICE agents have conveyed initial ICE descriptions and initial ICE responses, they will most likely continue gathering new local candidates as STUN, TURN, and other non-host candidate gathering mechanisms begin to yield results. Whenever an agent discovers such a new candidate, it will compute its priority, type, foundation, and component ID according to regular ICE procedures.

The new candidate is then checked for redundancy against the existing list of local candidates. If its transport address and base match those of an existing candidate, it will be considered redundant and will be ignored. This would often happen for server-reflexive candidates that match the host addresses they were obtained from (e.g., when the latter are public IPv4 addresses). Contrary to regular ICE, Trickle ICE agents will consider the new candidate redundant regardless of its priority.

Next, the agent "trickles" the newly discovered candidate(s) to the remote agent. The actual delivery of the new candidates is handled by a using protocol such as SIP or XMPP. Trickle ICE imposes no restrictions on the way this is done (e.g., some using protocols might choose not to trickle updates for server-reflexive candidates and instead rely on the discovery of peer-reflexive ones).

When candidates are trickled, the using protocol **MUST** deliver each candidate (and any end-of-candidates indication as described in Section 13) to the receiving Trickle ICE implementation exactly once and in the same order it was conveyed. If the using protocol provides any candidate retransmissions, they need to be hidden from the ICE implementation.

Also, candidate trickling needs to be correlated to a specific ICE session, so that if there is an ICE restart, any delayed updates for a previous session can be recognized as such and ignored by the receiving party. For example, using protocols that signal candidates via SDP might include a Username Fragment value in the corresponding a=candidate line, such as:

```
a=candidate:1 1 UDP 2130706431 2001:db8::1 5000 typ host ufrag 8hhY
```

Or, as another example, WebRTC implementations might include a Username Fragment in the JavaScript objects that represent candidates.

Note: The using protocol needs to provide a mechanism for both parties to indicate and agree on the ICE session in force (as identified by the Username Fragment and Password combination), so that they have a consistent view of which candidates are to be paired. This is especially important in the case of ICE restarts (see Section 15).

Note: A using protocol might prefer not to trickle server-reflexive candidates to entities that are known to be publicly accessible and where sending a direct STUN binding request is likely to reach the destination faster than the trickle update that travels through the signaling path.

10. Pairing Newly Gathered Local Candidates

As a Trickle ICE agent gathers local candidates, it needs to form candidate pairs; this works as described in the ICE specification [RFC8445], with the following provisos:

1. A Trickle ICE agent **MUST NOT** pair a local candidate until it has been trickled to the remote party.
2. Once the agent has conveyed the local candidate to the remote party, the agent checks if any remote candidates are currently known for this same stream and component. If not, the agent merely adds the new candidate to the list of local candidates (without pairing it).
3. Otherwise, if the agent has already learned of one or more remote candidates for this stream and component, it attempts to pair the new local candidate as described in the ICE specification [RFC8445].
4. If a newly formed pair has a local candidate whose type is server-reflexive, the agent **MUST** replace the local candidate with its base before completing the relevant redundancy tests.
5. The agent prunes redundant pairs by following the rules in Section 6.1.2.4 of [RFC8445] but checks existing pairs only if they have a state of Waiting or Frozen; this avoids removal of pairs for which connectivity checks are in flight (a state of In-Progress) or for which connectivity checks have already yielded a definitive result (a state of Succeeded or Failed).
6. If, after completing the relevant redundancy tests, the checklist where the pair is to be added already contains the maximum number of candidate pairs (100 by default as per [RFC8445]), the agent **SHOULD** discard any pairs in the Failed state to make room for the new pair. If there are no such pairs, the agent **SHOULD** discard a pair with a lower priority than the new pair in order to make room for the new pair, until the number of pairs is equal to the maximum number of pairs. This processing is consistent with Section 6.1.2.5 of [RFC8445].

11. Receiving Trickled Candidates

At any time during an ICE session, a Trickle ICE agent might receive new candidates from the remote agent, from which it will attempt to form a candidate pair; this works as described in the ICE specification [RFC8445], with the following provisos:

1. The agent checks if any local candidates are currently known for this same stream and component. If not, the agent merely adds the new candidate to the list of remote candidates (without pairing it).

2. Otherwise, if the agent has already gathered one or more local candidates for this stream and component, it attempts to pair the new remote candidate as described in the ICE specification [RFC8445].
3. If a newly formed pair has a local candidate whose type is server-reflexive, the agent **MUST** replace the local candidate with its base before completing the redundancy check in the next step.
4. The agent prunes redundant pairs as described below but checks existing pairs only if they have a state of Waiting or Frozen; this avoids removal of pairs for which connectivity checks are in flight (a state of In-Progress) or for which connectivity checks have already yielded a definitive result (a state of Succeeded or Failed).

   A. If the agent finds a redundancy between two pairs and one of those pairs contains a newly received remote candidate whose type is peer-reflexive, the agent **SHOULD** discard the pair containing that candidate, set the priority of the existing pair to the priority of the discarded pair, and re-sort the checklist. (This policy helps to eliminate problems with remote peer-reflexive candidates for which a STUN Binding request is received before signaling of the candidate is trickled to the receiving agent, such as a different view of pair priorities between the local agent and the remote agent, because the same candidate could be perceived as peer-reflexive by one agent and as server-reflexive by the other agent.)

   B. The agent then applies the rules defined in Section 6.1.2.4 of [RFC8445].

5. If, after completing the relevant redundancy tests, the checklist where the pair is to be added already contains the maximum number of candidate pairs (100 by default as per [RFC8445]), the agent **SHOULD** discard any pairs in the Failed state to make room for the new pair. If there are no such pairs, the agent **SHOULD** discard a pair with a lower priority than the new pair in order to make room for the new pair, until the number of pairs is equal to the maximum number of pairs. This processing is consistent with Section 6.1.2.5 of [RFC8445].

12. Inserting Trickled Candidate Pairs into a Checklist

After a local agent has trickled a candidate and formed a candidate pair from that local candidate (Section 9), or after a remote agent has received a trickled candidate and formed a candidate pair from that remote candidate (Section 11), a Trickle ICE agent adds the new candidate pair to a checklist as defined in this section.

As an aid to understanding the procedures defined in this section, consider the following tabular representation of all checklists in an agent (note that initially for one of the foundations, i.e., f5, there are no candidate pairs):

	f1	f2	f3	f4	f5
s1 (Audio.RTP)	F	F	F		
s2 (Audio.RTCP)	F	F	F	F	
s3 (Video.RTP)	F				

	f1	f2	f3	f4	f5
s4 (Video.RTCP)	F				

Table 1: Example of Checklist State

Each row in the table represents a component for a given data stream (e.g., s1 and s2 might be the RTP and RTP Control Protocol (RTCP) components for audio) and thus a single checklist in the checklist set. Each column represents one foundation. Each cell represents one candidate pair. In the tables shown in this section, "F" stands for "frozen", "W" stands for "waiting", and "S" stands for "succeeded"; in addition, "^^" is used to notate newly added candidate pairs.

When an agent commences ICE processing, in accordance with Section 6.1.2.6 of [RFC8445], for each foundation it will unfreeze the pair with the lowest component ID and, if the component IDs are equal, with the highest priority (this is the topmost candidate pair in every column). This initial state is shown in the following table.

	f1	f2	f3	f4	f5
s1 (Audio.RTP)	W	W	W		
s2 (Audio.RTCP)	F	F	F	W	
s3 (Video.RTP)	F				
s4 (Video.RTCP)	F				

Table 2: Initial Checklist State

Then, as the checks proceed (see Section 7.2.5.4 of [RFC8445]), for each pair that enters the Succeeded state (denoted here by "S"), the agent will unfreeze all pairs for all data streams with the same foundation (e.g., if the pair in column 1, row 1 succeeds then the agent will unfreeze the pairs in column 1, rows 2, 3, and 4).

	f1	f2	f3	f4	f5
s1 (Audio.RTP)	S	W	W		
s2 (Audio.RTCP)	W	F	F	W	
s3 (Video.RTP)	W				
s4 (Video.RTCP)	W				

Table 3: Checklist State with Succeeded Candidate Pair

Trickle ICE preserves all of these rules as they apply to "static" checklist sets. This implies that if a Trickle ICE agent were to begin connectivity checks with all of its pairs already present, the way that pair states change is indistinguishable from that of a regular ICE agent.

Of course, the major difference with Trickle ICE is that checklist sets can be dynamically updated because candidates can arrive after connectivity checks have started. When this happens, an agent sets the state of the newly formed pair as described below.

Rule 1: If the newly formed pair has the lowest component ID and, if the component IDs are equal, the highest priority of any candidate pair for this foundation (i.e., if it is the topmost pair in the column), set the state to Waiting. For example, this would be the case if the newly formed pair were placed in column 5, row 1. This rule is consistent with Section 6.1.2.6 of [RFC8445].

	f1	f2	f3	f4	f5
s1 (Audio.RTP)	S	W	W		^W^
s2 (Audio.RTCP)	W	F	F	W	
s3 (Video.RTP)	W				
s4 (Video.RTCP)	W				

Table 4: Checklist State with Newly Formed Pair, Rule 1

Rule 2: If there is at least one pair in the Succeeded state for this foundation, set the state to Waiting. For example, this would be the case if the pair in column 5, row 1 succeeded and the newly formed pair were placed in column 5, row 2. This rule is consistent with Section 7.2.5.3.3 of [RFC8445].

	f1	f2	f3	f4	f5
s1 (Audio.RTP)	S	W	W		S
s2 (Audio.RTCP)	W	F	F	W	^W^
s3 (Video.RTP)	W				
s4 (Video.RTCP)	W				

Table 5: Checklist State with Newly Formed Pair, Rule 2

Rule 3: In all other cases, set the state to Frozen. For example, this would be the case if the newly formed pair were placed in column 3, row 3.

	f1	f2	f3	f4	f5
s1 (Audio.RTP)	S	W	W		S
s2 (Audio.RTCP)	W	F	F	W	W
s3 (Video.RTP)	W		^F^		
s4 (Video.RTCP)	W				

Table 6: Checklist State with Newly Formed Pair, Rule 3

13. Generating an End-of-Candidates Indication

Once all candidate gathering is completed or expires for an ICE session associated with a specific data stream, the agent will generate an "end-of-candidates" indication for that session and convey it to the remote agent via the signaling channel. Although the exact form of the indication depends on the using protocol, the indication **MUST** specify the generation (Username Fragment and Password combination), so that an agent can correlate the end-of-candidates indication with a particular ICE session. The indication can be conveyed in the following ways:

- As part of an initiation request (which would typically be the case with the initial ICE description for half trickle)
- Along with the last candidate an agent can send for a stream
- As a standalone notification (e.g., after STUN Binding requests or TURN Allocate requests to a server time out and the agent is no longer actively gathering candidates)

Conveying an end-of-candidates indication in a timely manner is important in order to avoid ambiguities and speed up the conclusion of ICE processing. In particular:

- A controlled Trickle ICE agent **SHOULD** convey an end-of-candidates indication after it has completed gathering for a data stream, unless ICE processing terminates before the agent has had a chance to complete gathering.
- A controlling agent **MAY** conclude ICE processing prior to conveying end-of-candidates indications for all streams. However, it is **RECOMMENDED** for a controlling agent to convey end-of-candidates indications whenever possible for the sake of consistency and to keep middleboxes and controlled agents up-to-date on the state of ICE processing.

When conveying an end-of-candidates indication during trickling (rather than as a part of the initial ICE description or a response thereto), it is the responsibility of the using protocol to define methods for associating the indication with one or more specific data streams.

An agent **MAY** also choose to generate an end-of-candidates indication before candidate gathering has actually completed, if the agent determines that gathering has continued for more than an acceptable period of time. However, an agent **MUST NOT** convey any more candidates after it has conveyed an end-of-candidates indication.

When performing half trickle, an agent **SHOULD** convey an end-of-candidates indication together with its initial ICE description unless it is planning to potentially trickle additional candidates (e.g., in case the remote party turns out to support Trickle ICE).

After an agent conveys the end-of-candidates indication, it will update the state of the corresponding checklist as explained in Section 8. Past that point, an agent **MUST NOT** trickle any new candidates within this ICE session. Therefore, adding new candidates to the negotiation is possible only through an ICE restart (see Section 15).

This specification does not override regular ICE semantics for concluding ICE processing. Therefore, even if end-of-candidates indications are conveyed, an agent will still need to go through pair nomination. Also, if pairs have been nominated for components and data streams, ICE processing **MAY** still conclude even if end-of-candidates indications have not been received for all streams. In all cases, an agent **MUST NOT** trickle any new candidates within an ICE session after nomination of a candidate pair as described in Section 8.1.1 of [RFC8445].

14. Receiving an End-of-Candidates Indication

Receiving an end-of-candidates indication enables an agent to update checklist states and, in case valid pairs do not exist for every component in every data stream, determine that ICE processing has failed. It also enables an agent to speed up the conclusion of ICE processing when a candidate pair has been validated but uses a lower-preference transport such as TURN. In such situations, an implementation **MAY** choose to wait and see if higher-priority candidates are received; in this case, the end-of-candidates indication provides a notification that such candidates are not forthcoming.

When an agent receives an end-of-candidates indication for a specific data stream, it will update the state of the relevant checklist as per Section 8 (which might lead to some checklists being marked as Failed). If the checklist is still in the Running state after the update, the agent will note that an end-of-candidates indication has been received and take it into account in future updates to the checklist.

After an agent has received an end-of-candidates indication, it **MUST** ignore any newly received candidates for that data stream or data session.

15. Subsequent Exchanges and ICE Restarts

Before conveying an end-of-candidates indication, either agent **MAY** convey subsequent candidate information at any time allowed by the using protocol. When this happens, agents will use semantics from [RFC8445] (e.g., checking of the Username Fragment and Password combination) to determine whether or not the new candidate information requires an ICE restart.

If an ICE restart occurs, the agents can assume that Trickle ICE is still supported if support was determined previously; thus, they can engage in Trickle ICE behavior as they would in an initial exchange of ICE descriptions where support was determined through a capabilities discovery method.

16. Half Trickle

In half trickle, the initiator conveys the initial ICE description with a usable but not necessarily full generation of candidates. This ensures that the ICE description can be processed by a regular ICE responder and is mostly meant for use in cases where support for Trickle ICE cannot be confirmed prior to conveying the initial ICE description. The initial ICE description indicates support for Trickle ICE, so that the responder can respond with something less than a full generation of candidates and then trickle the rest. The initial ICE description for half trickle can contain an end-of-candidates indication, although this is not mandatory because if trickle support is confirmed, then the initiator can choose to trickle additional candidates before it conveys an end-of-candidates indication.

The half-trickle mechanism can be used in cases where there is no way for an agent to verify in advance whether a remote party supports Trickle ICE. Because the initial ICE description contains a full generation of candidates, it can thus be handled by a regular ICE agent, while still allowing a Trickle ICE agent to use the optimization defined in this specification. This prevents negotiation from failing in the former case while still giving roughly half the Trickle ICE benefits in the latter.

Use of half trickle is only necessary during an initial exchange of ICE descriptions. After both parties have received an ICE description from their peer, they can each reliably determine Trickle ICE support and use it for all subsequent exchanges (see Section 15).

In some instances, using half trickle might bring more than just half the improvement in terms of user experience. This can happen when an agent starts gathering candidates upon user-interface cues that the user will soon be initiating an interaction, such as activity on a keypad or the phone going off hook. This would mean that some or all of the candidate gathering could be completed before the agent actually needs to convey the candidate information. Because the responder will be able to trickle candidates, both agents will be able to start connectivity checks and complete ICE processing earlier than with regular ICE and potentially even as early as with full trickle.

However, such anticipation is not always possible. For example, a multipurpose user agent or a WebRTC web page where communication is a non-central feature (e.g., calling a support line in case of a problem with the main features) would not necessarily have a way of distinguishing between call intentions and other user activity. In such cases, using full trickle is most likely to result in an ideal user experience. Even so, using half trickle would be an improvement over regular ICE because it would result in a better experience for responders.

17. Preserving Candidate Order While Trickling

One important aspect of regular ICE is that connectivity checks for a specific foundation and component are attempted simultaneously by both agents, so that any firewalls or NATs fronting the agents would whitelist both endpoints and allow all except for the first ("suicide") packets to go through. This is also important to unfreezing candidates at the right time. While not crucial, preserving this behavior in Trickle ICE is likely to improve ICE performance.

To achieve this, when trickling candidates, agents **SHOULD** respect the order of components as reflected by their component IDs; that is, candidates for a given component **SHOULD NOT** be conveyed prior to candidates for a component with a lower ID number within the same foundation. In addition, candidates **SHOULD** be paired, following the procedures in Section 12, in the same order they are conveyed.

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

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

For this candidate information, the RTCP host candidate would not be conveyed prior to the RTP host candidate. Similarly, the RTP server-reflexive candidate would be conveyed together with or prior to the RTCP server-reflexive candidate.

18. Requirements for Using Protocols

In order to fully enable the use of Trickle ICE, this specification defines the following requirements for using protocols.

- A using protocol **SHOULD** provide a way for parties to advertise and discover support for Trickle ICE before an ICE session begins (see Section 3).
- A using protocol **MUST** provide methods for incrementally conveying (i.e., "trickling") additional candidates after conveying the initial ICE description (see Section 9).

- A using protocol **MUST** deliver each trickled candidate or end-of-candidates indication exactly once and in the same order it was conveyed (see Section 9).
- A using protocol **MUST** provide a mechanism for both parties to indicate and agree on the ICE session in force (see Section 9).
- A using protocol **MUST** provide a way for parties to communicate the end-of-candidates indication, which **MUST** specify the particular ICE session to which the indication applies (see Section 13).

19. IANA Considerations

IANA has registered the following ICE option in the "ICE Options" subregistry of the "Interactive Connectivity Establishment (ICE) registry", following the procedures defined in [RFC6336].

ICE Option: trickle

Contact: IESG <iesg@ietf.org>

Change controller: IESG

Description: An ICE option of 'trickle' indicates support for incremental communication of ICE candidates.

Reference: RFC 8838

20. Security Considerations

This specification inherits most of its semantics from [RFC8445], and as a result, all security considerations described there apply to Trickle ICE.

If the privacy implications of revealing host addresses on an endpoint device are a concern (see, for example, the discussion in [RFC8828] and in Section 19 of [RFC8445]), agents can generate ICE descriptions that contain no candidates and then only trickle candidates that do not reveal host addresses (e.g., relayed candidates).

21. References

21.1. Normative References

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <https://www.rfc-editor.org/info/rfc8445>.

21.2. Informative References

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

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

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

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

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

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

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

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <https://www.rfc-editor.org/info/rfc6120>.

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

[RFC8828] Uberti, J. and G. Shieh, "WebRTC IP Address Handling Requirements", RFC 8828, DOI 10.17487/RFC8828, January 2021, <https://www.rfc-editor.org/info/rfc8828>.

[RFC8840] Ivov, E., Stach, T., Marocco, E., and C. Holmberg, "A Session Initiation Protocol (SIP) Usage for Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (Trickle ICE)", RFC 8840, DOI 10.17487/RFC8840, January 2021, <https://www.rfc-editor.org/info/rfc8840>.

[XEP-0030] Hildebrand, J., Millard, P., Eatmon, R., and P. Saint-Andre, "XEP-0030: Service Discovery", XMPP Standards Foundation, XEP-0030, June 2008.

[XEP-0176] Beda, J., Ludwig, S., Saint-Andre, P., Hildebrand, J., Egan, S., and R. McQueen, "XEP-0176: Jingle ICE-UDP Transport Method", XMPP Standards Foundation, XEP-0176, June 2009.

Appendix A. Interaction with Regular ICE

The ICE protocol was designed to be flexible enough to work in and adapt to as many network environments as possible. Despite that flexibility, ICE as specified in [RFC8445] does not by itself support Trickle ICE. This section describes how trickling of candidates interacts with ICE.

[RFC8445] describes the conditions required to update checklists and timer states while an ICE agent is in the Running state. These conditions are verified upon transaction completion, and one of them stipulates that:

> if there is not a valid pair in the valid list for each component of the data stream associated with the checklist, the state of the checklist is set to Failed.

This could be a problem and cause ICE processing to fail prematurely in a number of scenarios. Consider the following case:

1. Alice and Bob are both located in different networks with Network Address Translation (NAT). Alice and Bob themselves have different addresses, but both networks use the same private internet block (e.g., the "20-bit block" 172.16/12 specified in [RFC1918]).
2. Alice conveys to Bob the candidate 172.16.0.1, which also happens to correspond to an existing host on Bob's network.
3. Bob creates a candidate pair from his host candidate and 172.16.0.1, puts this one pair into a checklist, and starts checks.
4. These checks reach the host at 172.16.0.1 in Bob's network, which responds with an ICMP "port unreachable" error; per [RFC8445], Bob marks the transaction as Failed.

At this point, the checklist only contains a Failed pair, and the valid list is empty. This causes the data stream and potentially all ICE processing to fail, even though Trickle ICE agents can subsequently convey candidates that could succeed.

A similar race condition would occur if the initial ICE description from Alice contains only candidates that can be determined as unreachable from any of the candidates that Bob has gathered (e.g., this would be the case if Bob's candidates only contain IPv4 addresses and the first candidate that he receives from Alice is an IPv6 one).

Another potential problem could arise when a non-Trickle ICE implementation initiates an interaction with a Trickle ICE implementation. Consider the following case:

1. Alice's client has a non-Trickle ICE implementation.
2. Bob's client has support for Trickle ICE.
3. Alice and Bob are behind NATs with address-dependent filtering [RFC4787].
4. Bob has two STUN servers, but one of them is currently unreachable.

After Bob's agent receives Alice's initial ICE description, it would immediately start connectivity checks. It would also start gathering candidates, which would take a long time because of the unreachable STUN server. By the time Bob's answer is ready and conveyed to Alice, Bob's connectivity checks might have failed: until Alice gets Bob's answer, she won't be able to start connectivity checks and punch holes in her NAT. The NAT would hence be filtering Bob's checks as originating from an unknown endpoint.

Appendix B. Interaction with ICE-Lite

The behavior of ICE-lite agents that are capable of Trickle ICE does not require any particular rules other than those already defined in this specification and [RFC8445]. This section is hence provided only for informational purposes.

An ICE-lite agent would generate candidate information as per [RFC8445] and would indicate support for Trickle ICE. Given that the candidate information will contain a full generation of candidates, it would also be accompanied by an end-of-candidates indication.

When performing full trickle, a full ICE implementation could convey the initial ICE description or response thereto with no candidates. After receiving a response that identifies the remote agent as an ICE-lite implementation, the initiator can choose to not trickle any additional candidates. The same is also true in the case when the ICE-lite agent initiates the interaction and the full ICE agent is the responder. In these cases, the connectivity checks would be enough for the ICE-lite implementation to discover all potentially useful candidates as peer-reflexive. The following example illustrates one such ICE session using SDP syntax:

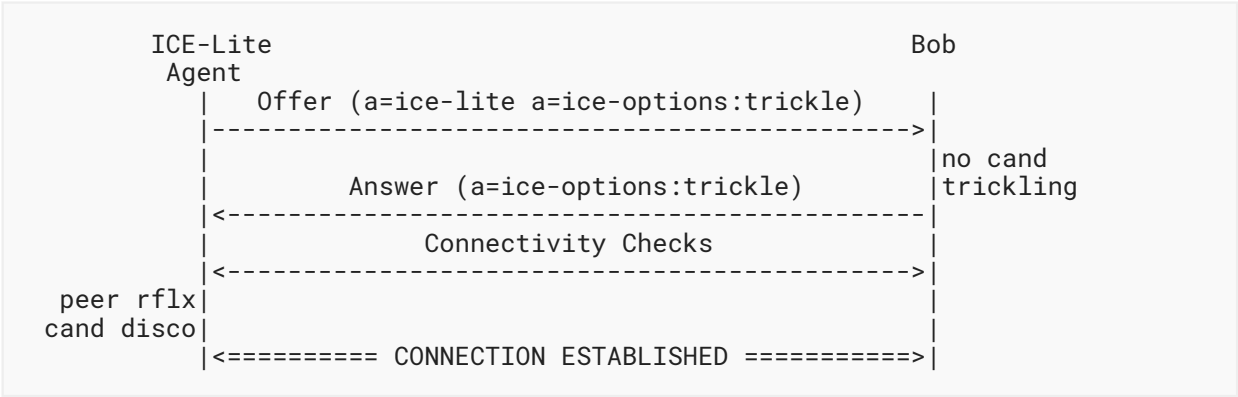

Figure 2: Example

In addition to reducing signaling traffic, this approach also removes the need to discover STUN Bindings or make TURN allocations, which can considerably lighten ICE processing.

Acknowledgements

The authors would like to thank Bernard Aboba, Flemming Andreasen, Rajmohan Banavi, Taylor Brandstetter, Philipp Hancke, Christer Holmberg, Ari Keränen, Paul Kyzivat, Jonathan Lennox, Enrico Marocco, Pal Martinsen, Nils Ohlmeier, Thomas Stach, Peter Thatcher, Martin Thomson, Brandon Williams, and Dale Worley for their reviews and suggestions on improving this document. Sarah Banks, Roni Even, and David Mandelberg completed OPSDIR, GenART, and security reviews, respectively. Thanks also to Ari Keränen and Peter Thatcher in their role as chairs and Ben Campbell in his role as responsible Area Director.

Authors' Addresses

Emil Ivov
8x8, Inc. / Jitsi
675 Creekside Way
Campbell, CA 95008
United States of America
Phone: +1 512 420 6968
Email: emcho@jitsi.org

Justin Uberti
Google
747 6th Street S
Kirkland, WA 98033
United States of America
Phone: +1 857 288 8888
Email: justin@uberti.name

Peter Saint-Andre
Mozilla
P.O. Box 787
Parker, CO 80134
United States of America
Phone: +1 720 256 6756
Email: stpeter@mozilla.com
URI: https://www.mozilla.com/